

# Appunti di Introduzione all’Intelligenza Artificiale Unipi - 3

## Parte

Raffaele Apetino

Marzo 2020

## Contents

<b>1</b>	<b>Introduzione - Machine Learning</b>	<b>4</b>
1.1	Un esempio concreto . . . . .	4
1.2	Quando usare il machine learning? . . . . .	4
1.3	Perché usare il machine learning? . . . . .	4
<b>2</b>	<b>Overview di un sistema ML</b>	<b>5</b>
2.0.1	Riconoscimento caratteri . . . . .	5
2.1	TASK: Supervised Learning . . . . .	6
2.2	TASK: UNsupervised Learning . . . . .	6
2.3	MODEL . . . . .	7
2.3.1	Esempi di Modelli . . . . .	7
2.4	LEARNING ALGORITHM . . . . .	7
2.5	VALIDATION - Generalizzazione . . . . .	8
<b>3</b>	<b>Concept Learning</b>	<b>8</b>
3.0.1	Esempio: Apprendimento Funzione Booleana . . . . .	9
3.1	Regole Congiuntive . . . . .	9
3.2	Problema: Enjoy Sport . . . . .	10
3.3	Rappresentare le Ipotesi . . . . .	10
3.4	TASK del Concept Learning . . . . .	10
3.5	Apprendimento Induttivo . . . . .	11
3.6	Qual é il numero di istanze, concetti e ipotesi di un problema? . . . . .	11
3.7	Da ordine generale a ordine specifico . . . . .	11
3.8	Algoritmo Find-S (Specific) . . . . .	12
3.8.1	Esempio su Enjoy Sport . . . . .	13
3.8.2	Proprietà di Find-S . . . . .	13
3.8.3	Aspetti negativi di Find-S . . . . .	13
3.9	Version Spaces . . . . .	13
3.10	Algoritmo List-Then-Eliminate . . . . .	14
3.10.1	Esempio di Version Space . . . . .	14
3.11	Algoritmo Candidate Elimination . . . . .	15
3.11.1	Esempio di Candidate Elimination . . . . .	16

<b>4 Inductive Bias</b>	<b>17</b>
4.1 Sistema di apprendimento Unbiased . . . . .	17
4.1.1 Il learning Unbiased è utile? . . . . .	17
4.2 Definizione Formale di Inductive Bias . . . . .	17
4.3 Sistemi Induttivi e Deduttivi equivalenti . . . . .	18
4.4 Tre (algoritmi) sistemi di apprendimento con Bias differenti . . . . .	18
<b>5 Modelli Lineari</b>	<b>19</b>
5.1 Esempio di regressione . . . . .	19
5.2 Modello di Regressione Lineare Semplice (Univariata) . . . . .	19
5.3 Costruzione della funzione via LMS . . . . .	20
5.3.1 Gradiente discendente . . . . .	22
5.4 Modello di Regressione Lineare Multivariato . . . . .	23
5.4.1 Notazione dei dati . . . . .	23
5.5 Hyperplane . . . . .	24
5.6 Gradiente Discendente Algoritmo . . . . .	24
5.7 Vantaggi dei modelli lineari . . . . .	25
5.8 Limitazioni . . . . .	25
5.8.1 Esempio . . . . .	25
5.9 Linear Basis Expansion . . . . .	26
5.9.1 Esempi . . . . .	26
5.9.2 Tradeoff per la complessità . . . . .	27
5.10 Regolarizzazione - Ridge Regression . . . . .	28
5.11 Limitazione delle funzioni a base fissa . . . . .	29
5.12 Classificazione con il modello lineare . . . . .	30
5.12.1 Visione geometrica dell'iperpiano . . . . .	30
5.12.2 Classificazione attraverso Decision Boundary lineare . . . . .	31
5.12.3 Esempio Terremoti/Esplosioni Nucleari . . . . .	31
5.12.4 Esempio Email Spam . . . . .	31
5.13 Il problema di apprendimento (per classificatori lineari) . . . . .	32
5.13.1 $\Delta w$ come regola della correzione dell'errore . . . . .	32
5.14 Classificazione dei pattern . . . . .	32
5.15 Congiunzioni nel caso di Modello Lineare . . . . .	33
5.15.1 Limitazioni . . . . .	33
5.16 Altri sistemi di apprendimento per la classificazione . . . . .	34
5.17 Conclusioni sui Modelli Lineari . . . . .	34
<b>6 Alberi di decisione</b>	<b>35</b>
6.1 Problema del PlayTennis . . . . .	35
6.2 (Alg. ID3) Induzione Top-Down sugli alberi di decisione . . . . .	36
6.3 Entropia: Come scegliere il miglior Attributo . . . . .	36
6.4 Information Gain . . . . .	37
6.4.1 Esempio . . . . .	38
6.5 Problemi con Information Gain . . . . .	38
6.6 Gain Ratio . . . . .	39
6.7 Considerazioni sulla ricerca nello Spazio delle Ipotesi (in DT Learning) . . . . .	39
6.8 Inductive Bias in DT Learning . . . . .	39
6.9 Problemi con gli alberi di decisione . . . . .	40
6.9.1 Evitare l'Overfitting . . . . .	41
6.9.2 Potatura con errore ridotto . . . . .	41
6.9.3 Regola della post-potatura . . . . .	41
6.10 Problema: Valori continui degli attributi . . . . .	42

6.11 Problema: Dati di training incompleti . . . . .	42
6.12 Problema: attributi con costi differenti . . . . .	42
6.13 Visione Geometrica (Decision Boundaries dei DT) . . . . .	43
6.14 Conclusioni sugli alberi di decisione . . . . .	43
<b>7 Validation</b>	<b>43</b>
7.1 Obiettivi Validazione . . . . .	44
7.2 Holdout Cross-Validation . . . . .	44
7.3 Meta-Algoritmo per l'utilizzo del Data Set . . . . .	44
7.3.1 Esempio . . . . .	45
7.3.2 Esempio perché separare VL e TS . . . . .	45
7.4 K-fold cross validation . . . . .	46
7.5 Esempio di Model Selection e Assessment . . . . .	46
7.6 Comportamento tipico di un algoritmo di apprendimento . . . . .	47
7.7 Statistical Learning Theory . . . . .	48
7.8 Teoria di Vapnik-Chervonenkis + SLT . . . . .	48
7.9 Structural Risk Minimization . . . . .	49
7.10 Conclusioni . . . . .	49
<b>8 Support Vector Machine</b>	<b>49</b>
8.0.1 Obiettivi utilizzo SVM . . . . .	49
8.1 Maximum Margin Classifier (controllo complessità) . . . . .	50
8.2 Rappresentazione canonica dell'iperpiano e Support Vector . . . . .	51
8.3 Verso l'ottimizzazione del margine . . . . .	52
8.4 Problema di ottimizzazione quadratico . . . . .	52
8.5 Nuovo classificatore $h(x)$ (Problema Duale) . . . . .	52
8.6 Margine Soft . . . . .	53
8.7 Mapping per Spazi Dimensionali Ampi . . . . .	54
8.7.1 Esempi di Kernel . . . . .	55
8.8 Riassunto procedimento SVM con Kernel Fun. . . . .	55
8.9 Utilizzare bene SVM . . . . .	55
<b>9 K-Nearest Neighbors</b>	<b>56</b>
9.1 1-Nearest Neighbor . . . . .	56
9.2 K-NN . . . . .	57
9.3 Considerazioni su K-nn . . . . .	58
9.4 Distance Based methods . . . . .	58
<b>10 Unsupervised Learning (ripasso)</b>	<b>58</b>
10.1 K-means . . . . .	58
10.2 Limitazioni K-means . . . . .	59
10.3 Preprocessing dei dati . . . . .	60
<b>11 Altri Task del Machine Learning</b>	<b>60</b>
<b>12 Altri Modelli del Machine Learning</b>	<b>61</b>
12.1 Reti Neurali . . . . .	61
12.2 Deep Learning . . . . .	61

# 1 Introduzione - Machine Learning

Il problema dell'apprendimento è uno dei problemi principali dell'intelligenza sia artificiale che biologica. L'apprendimento automatico (Machine Learning) è emerso come un'area di ricerca che combina gli obiettivi della creazione di macchine in grado di apprendere e strumenti statistici/adattivi. Perché le macchine dovrebbero imparare da sole? Poiché c'è una crescente necessità di analizzare dati empirici difficili da gestire con la normale programmazione (paradigma DATA-DRIVEN). Il compito dell'apprendimento automatico è la creazione di sistemi intelligenti adattivi, in grado di analizzare grandi quantità di dati.

## 1.1 Un esempio concreto

Esempi concreti di applicazione di ML possono essere la classificazione delle email spam oppure il riconoscimento di caratteri scritti a mano, dei visi o del parlato. Non ci sono regole o conoscenza pregressa per trovare la soluzione, ma diventa più facile avendo una fonte di esperienza per apprendere (dati di cui conosciamo già il risultato). Ad esempio nel riconoscimento facciale si combinano reti neurali e altri approcci di ML partendo da milioni di immagini facciali appartenenti a diversi individui.

## 1.2 Quando usare il machine learning?

Il machine learning è una grande opportunità ma deve essere controllato. Usiamo l'apprendimento automatico quando:

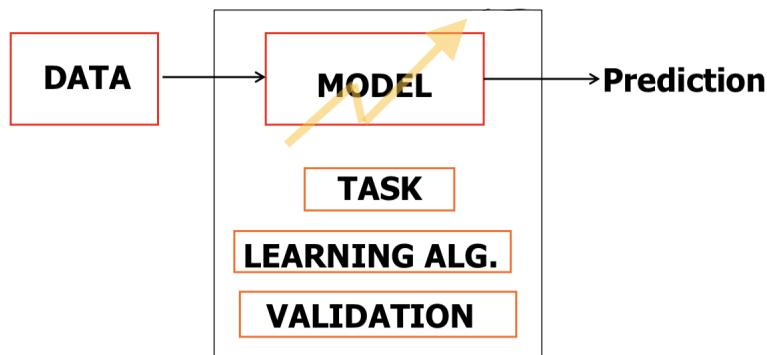
- non abbiamo conoscenza per spiegare il fenomeno oppure è difficile formalizzarlo
- abbiamo dati incerti, rumorosi o incompleti, che ostacolano la formalizzazione di soluzioni
- ci troviamo in ambienti dinamici non conosciuti in precedenza

Abbiamo però anche dei requisiti per poter applicare ML ai nostri problemi. È necessaria una fonte di esperienza formativa, ma spesso è difficile raccogliere molti dati rappresentativi (basti pensare alla raccolta di dati riguardanti una malattia rara). Dobbiamo anche considerare una tolleranza sulla precisione dei dati (se accettiamo di dare pochi dati in input, o poco significativi, dobbiamo anche tollerare una certa percentuale di errore, che però in alcuni casi non può essere accettata).

## 1.3 Perché usare il machine learning?

È una opportunità per conoscere nuovi paradigmi informatici con un approccio differente dalla programmazione standard, algoritmica e dall'IA classica. Serve per trovare soluzioni approssimate a problemi molto difficili. Non è una metodologia approssimativa, è un approccio rigoroso per trovare funzioni approssimative per affrontare problemi complessi.

## 2 Overview di un sistema ML

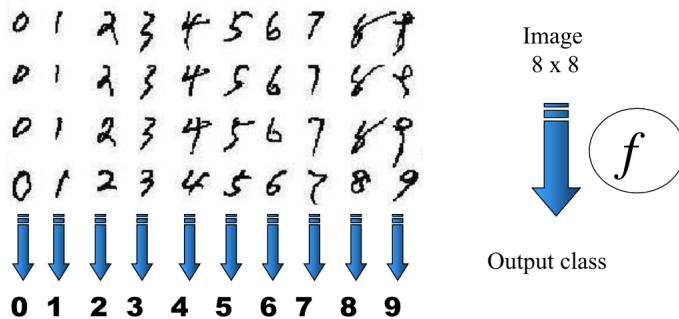


- DATA: i dati li ricaviamo dalle osservazioni del mondo
- MODEL: i dati passano attraverso un modello. Il modello non è fissato: ha dei parametri liberi, modificabili per fornire le predizioni in modo che siano quelle che vorremmo che fossero fornite.
  - TASK: Supervised o Unsupervised Learning
  - LEARNING ALGORITHM: algoritmo che cerca la soluzione in uno spazio di stati
  - VALIDATION: valutazione statistica del modello (quanto riesce ad essere accurato sui dati futuri).
- PREDICTION: è l'output restituito dal modello

"Apprendere" viene visto come una approssimazione di una funzione sconosciuta ricavata dagli esempi.

### 2.0.1 Riconoscimento caratteri

Un esempio "pilota" che possiamo fare è il riconoscimento di caratteri scritti a mano. Il problema è racchiuso nel riconoscere una funzione che passa per un insieme di punti. Come input abbiamo una collezione di immagini di caratteri scritti a mano. Il problema è costruire un modello che riceve in input queste matrici di 8x8 pixel e come output restituire la cifra riconosciuta.



Non sappiamo quale sia la funzione, perché in quel caso la scriveremmo direttamente in un algoritmo. Cerchiamo quindi di approssimare una funzione facendo restituire come output una classificazione della matrice di pixel. Il problema di classificazione è dato dalla formalizzazione della soluzione esatta, poiché ci possiamo trovare con dati rumorosi o ambigui. Risulta però facile raccogliere collezioni

di esempi etichettati, cioè esempi con soluzioni associate.

Generalizzando il problema possiamo usare la tecnica del Supervised Learning. In particolare abbiamo in input uno spazio contentente "x" dati etichettati, dobbiamo costruire una funzione generale a partire dagli esempi e come output dare una categoria o valori reali.

## 2.1 TASK: Supervised Learning

Il Supervised Learning è una tecnica di apprendimento automatico che mira a istruire un sistema informatico, in modo da consentirgli di elaborare automaticamente previsioni sui valori di uscita di un sistema rispetto ad un input sulla base di una serie di esempi ideali, costituiti da coppie di input e di output, che gli vengono inizialmente forniti.

Quindi vengono dati esempi di training nella forma di coppie  $\langle \text{input}, \text{output} \rangle = \langle x, d \rangle$  (esempi etichettati) per la creazione di una funzione sconosciuta  $f$ . Definiamo il valore target come il valore " $d$ " che vogliamo ottenere come output (e che ci viene dato tramite gli esempi).

Bisogna trovare una buona approssimazione di  $f$ , cioè una ipotesi  $h$  che può essere usata per predire l'output sui dati sconosciuti  $x'$ .

L'obiettivo è dare in output una etichetta numerica o la classificazione del dato.

- Classificazione: la funzione a valori discreti  $f(x)$  restituisce la presunta classe corretta per  $x$ .
- Regressione: consiste nell'approssimare a valori reali la funzione target.

Entrambi sono compiti di approssimazione di funzione.

## 2.2 TASK: UNsupervised Learning

L'apprendimento non supervisionato è una tecnica di apprendimento automatico che consiste nel fornire al sistema informatico una serie di input (esperienza del sistema) che egli riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi. Abbiamo quindi un TR set che è un insieme di dati non etichettati, il compito è quello di raggruppare i dati in insiemi consistenti. I principali algoritmi sono:

- Clustering: raggruppamento di elementi omogenei in un insieme di dati (cluster) identificando un centroide.
- Dimensionality reduction / Visualization / Preprocessing
- Modeling the data density

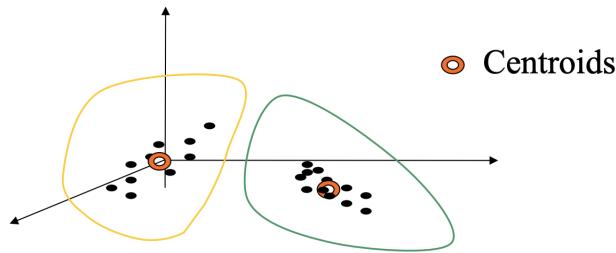


Figure 1: Esempio di clustering

## 2.3 MODEL

Il compito è quello di catturare e descrivere le relazioni tra i dati sulla base del TASK. Il modello definisce la classe delle funzioni che la macchina può implementare (cioè definisce lo spazio delle ipotesi  $H$ ).

Concetti utili:

- Training examples: fanno parte del Supervised Learning ed è un insieme di esempi nella forma  $(x, f(x))$  dove  $x$  è un vettore di caratteristiche e  $f(x)$  è il valore obiettivo.
- Target function: la funzione obiettivo  $f$  il più possibile corretta
- Hypothesis: è una funzione  $h$  che è ritenuta di essere simile a  $f$ . La funzione  $h$  è data in un linguaggio capace di esprimere le relazioni tra i dati.
- Hypotheses space: è lo spazio di tutte le ipotesi che possono essere output dell'algoritmo. È dove cerchiamo la funzione  $h$ .

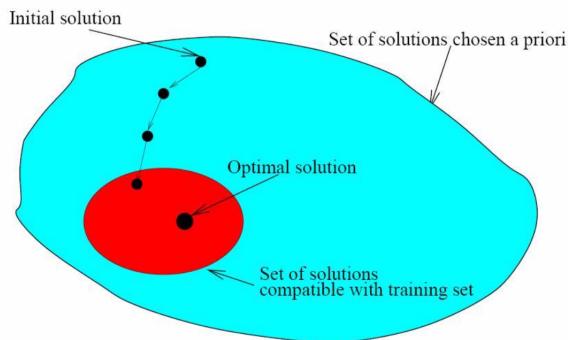
I linguaggi per le ipotesi  $h$  devono saper esprimere modelli di ML: ad esempio la logica del primo ordine, equazioni numeriche oppure calcolo delle probabilità.

### 2.3.1 Esempi di Modelli

- Modelli Lineari: la rappresentazione dello spazio delle ipotesi  $H$  definisce uno spazio continuo parametrizzato di potenziali ipotesi. Come parametro abbiamo  $w$ , ed ogni assegnamento di  $w$  è una ipotesi differente. (es:  $h_w(x) = w_1x + w_0$ )
- Regole simboliche: lo spazio delle ipotesi è basato su rappresentazioni discrete, sono possibili differenti regole come ad esempio: if  $(x_1=0 \text{ and } x_2=1)$  then  $h(x)=1$  else  $h(x)=0$
- Modelli probabilistici: stimare  $p(x,y)$
- Approcci basati sull'istanza: confronta le nuove istanze del problema con le istanze osservate durante l'addestramento, che sono state memorizzate. Si chiama basato sull'istanza perché costruisce ipotesi direttamente dalle istanze di addestramento stesse.

## 2.4 LEARNING ALGORITHM

Gli algoritmi di apprendimento si basano su DATA, TASK e MODEL. L'euristica consiste nella ricerca delle migliori ipotesi nello spazio delle ipotesi  $H$ , cioè la miglior approssimazione della funzione obiettivo  $f$  sconosciuta (ad esempio i parametri liberi del modello vengono adattati al compito da svolgere: il miglior  $w$  nel modello lineare, la miglior regola per regole simboliche, ecc...).  $H$  potrebbe non coincidere con l'insieme di tutte le possibili funzioni e la ricerca può non risultare esaustiva, per questo bisogna fare assunzioni (vedremo in seguito il ruolo dell'Inductive Bias).



## 2.5 VALIDATION - Generalizzazione

L'apprendimento può essere quindi definito come la ricerca una buona funzione, in uno spazio di funzioni restituito da dati conosciuti. "Buona funzione" considerando l'errore di generalizzazione (o capacità di generalizzazione) cioè quanto accuratamente il modello prevede nuovi campioni di dati. La generalizzazione è cruciale in ML → strumenti corretti di ML.

- Fase di apprendimento (training, fitting): costruire il modello dai dati di training conosciuti (Training Set).
- Fase predittiva (test): si applica la funzione costruita a un nuovo esempio. Si prende in input  $x$ , si computa la risposta dal modello e confrontiamo l'output con il nostro target (che il modello non ha mai visto). Si esegue una valutazione dell'ipotesi predittiva, ad esempio la valutazione della capacità di generalizzazione (valutazione statistica del modello, quanto riesce ad essere accurato sui dati futuri). È bene notare che essere performanti in ML significa predire accuratamente, cioè la performance è stimata dagli errori effettuati calcolando la funzione sul Test Set (che è diverso dal Training set!).

## 3 Concept Learning

Il Concept Learning anche noto come apprendimento di categoria è definito come "la ricerca di attributi che possono essere utilizzati per distinguere i modelli dai non modelli di varie categorie". Più semplicemente, i concetti sono le categorie che ci aiutano a classificare oggetti, eventi o idee, basandoci sul fatto che ogni oggetto, evento o idea ha un insieme di caratteristiche rilevanti comuni. In un compito di Concept Learning, una macchina apprende come classificare degli oggetti mostrandole prima una serie di esempi associati alle loro etichette di classe. La macchina semplifica quello che ha osservato "condensandolo" sotto forma di un esempio. Questa versione semplificata di ciò che è stato appreso viene quindi applicata a esempi futuri.

Lavoriamo in uno spazio discreto strutturato come lo spazio delle ipotesi, useremo come rappresentazione la congiunzione di letterali e vedremo algoritmi come Find-S ed Candidate Elimination. Praticamente in questa sezione andremo ad vedere in cosa consiste la classificazione del Supervised Learning, cioè come la funzione  $f(x)$  restituisce la presunta classe corretta per  $x$ .

Concept Learning: dedurre una funzione booleana (con dominio  $X$  e codominio  $\{t,f\}$ ) da esempi di allenamento positivi e negativi (dove  $X$  è lo spazio che contiene le istanze)

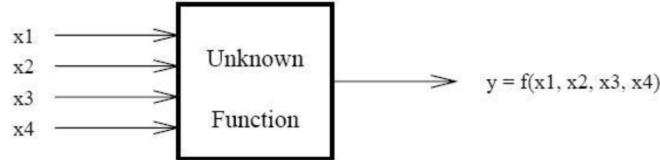
Un Training Example è definito come coppia  $\langle x, c(x) \rangle$  (le coppie sono contenute nel Training Set). L'ipotesi  $h$ :  $X \rightarrow \{0, 1\}$  soddisfa  $x$  se  $h(x)=1$ .

Una ipotesi  $h$  è consistente con un esempio di allenamento  $\langle x, c(x) \rangle$  se  $h(x)=c(x)$  con  $x$  appartenente a  $X$ . È consistente anche con  $D^1$  se  $h(x)=c(x)$  per ogni esempio di allenamento  $\langle x, c(x) \rangle$  in  $D$ .

---

<sup>1</sup>è sempre il Training Set insieme degli esempi

### 3.0.1 Esempio: Apprendimento Funzione Booleana



Example	$x_1$	$x_2$	$x_3$	$x_4$	$y$
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Questo è un problema mal posto poiché potremmo violare l'esistenza, unicità o stabilità della soluzione. Come possiamo vedere dall'immagine abbiamo in input 4 variabili e in output un singolo valore. Nella tabella troviamo il nostro training set (D) e il nostro compito è quello di trovare la funzione che dati in input quei valori restituisce l'output definito dalla tabella. Ci sono  $2^{2^4}$  possibili funzioni booleane per 4 ingressi, questo perché abbiamo  $2^4$  possibili ingressi e per ognuno possiamo rispondere 0 o 1. Non possiamo sapere quale sia la funzione corretta fino a che non abbiamo visto tutte le possibili coppie di input/output. Dopo aver analizzato i nostri 7 esempi ci rimangono ancora  $2^9$  possibilità. Nel caso generale per input/output binari la formula è  $|H| = 2^{\text{numero istanze}} = 2^n$  dove  $|H|$  è la dimensione dello spazio delle ipotesi ed  $n$  è la dimensione dell'input.

Lavoreremo con uno spazio delle ipotesi ristretto: partiamo scegliendo uno spazio delle ipotesi  $H$  che è considerevolmente più piccolo dello spazio di tutte le possibili funzioni.

Vedremo:

- proposizioni formate da solo "and" (regole congiuntive) in uno spazio di ipotesi  $H$  finito e discreto
- funzioni lineari, in uno spazio di ipotesi  $H$  continuo e infinito

### 3.1 Regole Congiuntive

Quante h diverse possiamo avere? Cioè quante semplici regole congiuntive?

Nel caso generale:

- Letterali positivi: ad esempio  $h_1 = l_2$ ,  $h_2 = (l_1 \wedge l_2)$ ,  $h_3 = \text{true}$  ecc... sono semplici regole congiuntive. Abbiamo ridotto lo spazio delle ipotesi così da ottenere  $|H| = 2^n$  basti immaginare  $l_i$  come una stringa di bit di lunghezza n.
- Letterali: (di cui fa parte anche il  $\text{not}(l_i)$ ) ottenendo uno spazio delle ipotesi di dimensione  $3^n + 1$

Vedremo come organizzare e cercare in modo efficiente attraverso uno spazio di ipotesi, sfruttando algoritmi per un insieme di ipotesi molto limitato semplificandoci il lavoro usando dati non rumorosi.

### 3.2 Problema: Enjoy Sport

Concept: "giorni in cui piace fare sport acquatici"

Task: predire il valore di "Enjoy Sport" per un giorno arbitrario in base ai valori degli attributi.

Sky	Temp	Humid	Wind	Water	Forecast	Enjoy Sport?
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

Una riga della tabella è una istanza.

### 3.3 Rappresentare le Ipotesi

L'ipotesi  $h$  è una congiunzione di vincoli sugli attributi. Ogni vincolo può essere:

- Uno specifico valore: ad esempio Water = Warm
- Un valore irrilevante: ad esempio Water = ?
- Nessun valore consentito (ipotesi nulla): ad esempio Water = 0

Esempio di ipotesi  $h$ :

Sky	Temp	Humid	Wind	Water	Forecast
Sunny	?	?	Strong	?	Same

corrispondente alla funzione booleana

$$Sky = Sunny \wedge Wind = Strong \wedge Forecast = Same$$

L'ipotesi più specifica (risponde sempre false) e più generale (risponde sempre true) sono rispettivamente

Sky	Temp	Humid	Wind	Water	Forecast
0	0	0	0	0	0 //specifico
?	?	?	?	?	? //generale

### 3.4 TASK del Concept Learning

Vengono dati:

- Le istanze  $X$ : i possibili giorni vengono descritti da attributi Sky, Temp, Humid, Wind, Water e Forecast.
- Funzione Target:  $c: EnjoySport : X \rightarrow \{0, 1\}$ .
- Spazio delle ipotesi ( $H$ ): insieme finito di coniunzioni di letterali
- Esempi di allenamento ( $D$ ): esempi positivi e negativi:  $\langle x_1, c(x_1) \rangle, \dots, \langle x_n, c(x_n) \rangle$

Bisogna trovare:

- Una ipotesi  $h$  in  $H$  tale che  $h(x)=c(x)$  per tutti gli  $x$  in  $X$

L'apprendimento sta nella ricerca nello spazio delle ipotesi  $H$ .

### 3.5 Apprendimento Induttivo

Ipotesi Apprendimento Induttivo: "Qualsiasi ipotesi trovata per approssimare la funzione target sugli esempi di allenamento, approssimerà anche la funzione target sugli esempi non osservati". Ma purtroppo non è detto che il Training Set sia corretto (dati rumorosi, ...)

Quindi  $h(x)=c(x)$  per ogni  $x$  in  $D$  (funzione consistente con il training set) ma  $h(x)=c(x)$  per ogni  $x$  in  $X$ ? Problema fondamentale del ML.

### 3.6 Qual é il numero di istanze, concetti e ipotesi di un problema?

- Sky: Sunny, Cloudy, Rainy
- Temp: Warm, Cold
- Humid: Normal, High
- Wind: Strong, Weak
- Water: Warm, Cold
- Forecast: Same, Change

La scelta della rappresentazione di  $H$  determina lo spazio di ricerca!

Numero di possibili istanze:  $3*2*2*2*2*2 = 96$

Numero di concetti distinti:  $2^{96} = 2^{\text{numero istanze}}$

Numero di ipotesi sintatticamente distinte:  $5*4*4*4*4*4$  (poiché per ogni attributo devo aggiungere 0 o ?)

Numero di ipotesi semanticamente distinte:  $1+4*3*3*3*3*3$  (poiché le ipotesi con almeno uno 0 sono equivalenti a false, i valori sono in and!)

Strutturare uno spazio di ricerca può aiutare a cercare in modo piú efficiente.

### 3.7 Da ordine generale a ordine specifico

Consideriamo due ipotesi

1.  $h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$
2.  $h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$

L'insieme di istanze coperte da  $h_1$  e da  $h_2$  sono differenti, infatti  $h_2$  impone meno vincoli rispetto a  $h_1$  e quindi classifica più istanze x positive ( $h_2(x)=1$ ).

Siano  $h_j$  e  $h_k$  funzioni booleane definite su  $X$ .  $h_j$  é più generale o equivalente a  $h_k$  ( $h_j \geq h_k$ ) se solo se  $\forall x \in X : [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$

Possiamo sfruttare questo ordine parziale per organizzare più efficientemente la nostra ricerca in  $H$ .

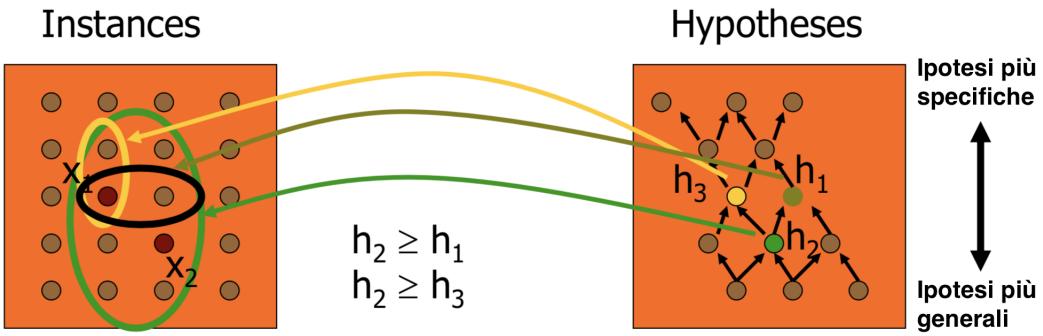


Figure 2: Partial Order

Istanze:

- $x_1 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Same} >$
- $x_2 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Light}, \text{Warm}, \text{Same} >$

Ipotesi:

- $h_1 = < \text{Sunny}, ?, ?, \text{Strong}, ?, ?, ? >$
- $h_2 = < \text{Sunny}, ?, ?, ?, ?, ?, ? >$
- $h_3 = < \text{Sunny}, ?, ?, ?, \text{Cool}, ? >$

### 3.8 Algoritmo Find-S (Specific)

L'algoritmo Find-S sfrutta l'ordine parziale di gestione per cercare in modo efficiente una  $h$  consistente (senza elencare esplicitamente ogni  $h$  in  $H$ ).

1. Inizializza  $h$  con l'ipotesi più specifica nello spazio delle ipotesi  $H$  ( $h_0 = < 0, 0, 0, 0, 0, 0 >$ )
2. Prendo il prossimo esempio di training:

- Per ogni attributo  $a_i$  dell'ipotesi  $h$ , se  $a_i$  è soddisfatto in  $h$  da  $x$  allora non fare nulla (quindi se l'esempio è negativo non ci sono cambiamenti sull'ipotesi), altrimenti se  $a_i$  NON è soddisfatto e l'esempio è positivo, sostituisce  $a_i$  con il prossimo vincolo più generale soddisfatto da  $x$ .  
Ripeto il passaggio per ogni esempio positivo.

3. Viene dato in output l'ipotesi  $h$

L'algoritmo parte dall'ipotesi più specifica, poi scende in maniera conservativa (generalizza il minimo possibile) in modo da rimanere consistente sui positivi, e di conseguenza contemporaneamente sui negativi. Trovo un'ipotesi che copre tutti i positivi e tutti i negativi. In questo modo evito di esplorare tutto  $H$ , scorrendo solo una volta il training set, perché si segue il Partial Order.

### 3.8.1 Esempio su Enjoy Sport

1.  $h_0 = \langle 0, 0, 0, 0, 0, 0 \rangle$
2.  $x_1 = \langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$  questo esempio è positivo<sup>2</sup> e  $h_0$  non è soddisfatta perché False. Quindi devo prendere i vincoli più generali da  $x_1$  e metterli in  $h$ , in questo caso sono le singole istanze poiché tutte più generali di 0.
3.  $h_1 = \langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$
4.  $x_2 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} \rangle$  esempio positivo, notiamo che High è in contrasto con Normal. Quindi cambio quell'attributo con uno più generale, in questo caso è "?".
5.  $h_2 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$
6.  $x_3 = \langle \text{Rainy}, \text{Cold}, \text{High}, \text{Strong}, \text{Warm}, \text{Change} \rangle$  esempio è negativo quindi non faccio nulla
7.  $h_3 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$
8.  $x_4 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Change} \rangle$  esempio positivo
9.  $h_3 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? \rangle$

### 3.8.2 Proprietà di Find-S

Lo spazio delle ipotesi è rappresentato come congiunzione di attributi (molto limitativo), l'algoritmo darà in output l'ipotesi più specifica, nello spazio  $H$ , che è consistente con gli esempi positivi del training set. L'ipotesi di output  $h$  sarà anche consistente con gli esempi negativi, a condizione che il concetto target sia contenuto in  $H$ . Sceglio l'ipotesi più specifica poiché nel caso in cui ci fossero diverse ipotesi consistenti con gli esempi di training, Find-S trova la più specifica.

### 3.8.3 Aspetti negativi di Find-S

Non so se il sistema di apprendimento converga con il concetto target, nel senso che non è in grado di determinare se ha trovato l'unica ipotesi coerente con gli esempi di allenamento. Inoltre non so quando i dati di allenamento sono incoerenti, in quanto ignora gli esempi di allenamento negativi → nessuna tolleranza al rumore sui dati!

## 3.9 Version Spaces

Find-S fornisce una singola ipotesi da  $H$  che è coerente con gli esempi di allenamento, questa è solo una delle tante ipotesi da  $H$  che potrebbero adattarsi ugualmente bene ai dati di allenamento. L'idea è quindi quella di dare in output l'insieme di tutte le possibili  $h$  consistenti con  $D$ .

Il Version Space  $VS_{H,D}$  rispetto allo spazio delle ipotesi  $H$  e il training set  $D$ , è il sottoinsieme delle ipotesi prese da  $h$  coerenti con tutti gli esempi di training:

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

dove  $\text{Consistent}(h, D) = \forall \langle x, c(x) \rangle \in D \mid h(x) = c(x)$

Si dice che un esempio  $x$  soddisfa l'ipotesi  $h$  quando  $h(x)=1$ , indipendentemente dal fatto che  $x$  sia un esempio positivo o negativo del concetto target. Tuttavia, se un tale esempio sia coerente con  $h$  dipende dal concetto target e, in particolare, se  $h(x)=c(x)$ .

---

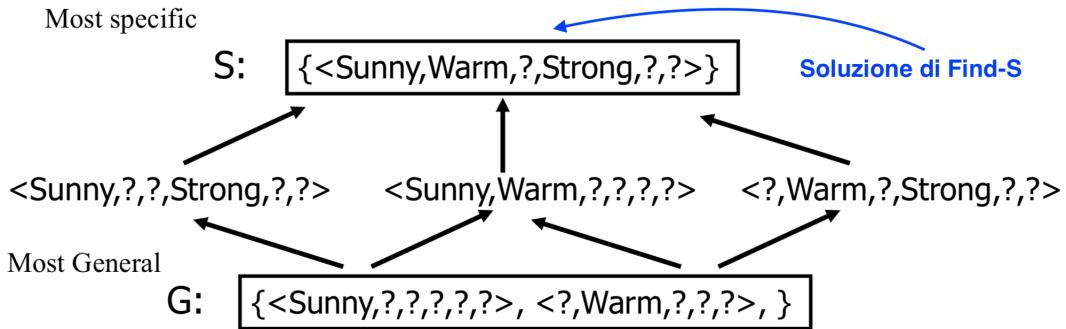
<sup>2</sup>l'esempio è preso dalla tabella a pagina 10

### 3.10 Algoritmo List-Then-Eliminate

L'algoritmo List-Then-Eliminate inizializza il Version Space con tutte le ipotesi contenute in  $H$  eliminando volta per volta tutte le ipotesi che non sono consistenti con gli esempi di training. Il Version Space si riduce quindi man mano che si osservano più esempi, fino a quando idealmente rimane solo un'ipotesi coerente con tutti gli esempi osservati, presumibilmente, questo è il concetto target desiderato. Se non sono disponibili dati sufficienti per restringere lo spazio l'algoritmo può generare l'intero insieme di ipotesi coerenti con i dati osservati. L'algoritmo List-Then-Eliminate può essere applicato ogni volta che lo spazio di ipotesi  $H$  è finito. Ha molti vantaggi, tra cui il fatto che è garantito il risultato di tutte le ipotesi coerenti con i dati di allenamento. Sfortunatamente, richiede un elenco esaustivo di tutte le ipotesi in  $H$ , un requisito non realistico per tutti gli spazi di ipotesi tranne quelli più banali.

1. Iniziamo impostando il nostro Version Space con una lista contenente ogni ipotesi in  $H$
2. Per ogni esempio di allenamento  $\langle x, c(x) \rangle$  rimuoviamo dal Version Space ogni ipotesi che è inconsistente con gli esempi di allenamento cioè  $h(x) \neq c(x)$
3. Viene dato in output la lista delle ipotesi ora contenute nel Version Space

#### 3.10.1 Esempio di Version Space



$$\begin{aligned}
 x_1 &= \langle \text{Sunny Warm Normal Strong Warm Same} \rangle + \\
 x_2 &= \langle \text{Sunny Warm High Strong Warm Same} \rangle + \\
 x_3 &= \langle \text{Rainy Cold High Strong Warm Change} \rangle - \\
 x_4 &= \langle \text{Sunny Warm High Strong Cool Change} \rangle +
 \end{aligned}$$

Il Version Space dell'algoritmo viene rappresentato solo dal suo membro più generale G e dal più specifico S (che è la soluzione di Find-S!).

Teorema: ogni membro del Version Space si trova tra:

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq h \geq s)\}$$

dove  $x \geq y$  significa x è più generale o uguale a y.

### 3.11 Algoritmo Candidate Elimination

L'algoritmo di Candidate Elimination calcola il Version Space contenente tutte le ipotesi di  $H$  che sono coerenti con una sequenza osservata di esempi di addestramento. L'algoritmo calcola il Version Space senza elencarne esplicitamente tutti i suoi membri, ciò si ottiene utilizzando l'ordinamento più generale che parziale e mantenendo una rappresentazione compatta dell'insieme di ipotesi coerenti.

Inizia inizializzando il Version Space sull'insieme di tutte le ipotesi in  $H$ , cioè inizializzando il set di limiti  $G$  per contenere l'ipotesi più generale in  $H$   $G_0 = <?, ?, ?, ?, ?, ? >$  e inizializzando il set di limiti  $S$  per contenere l'ipotesi più specifica  $S_0 = <0, 0, 0, 0, 0, 0>$ . Questi due insiemi di limiti delimitano l'intero spazio delle ipotesi, poiché ogni altra ipotesi in  $H$  è sia più generale di  $S_0$  sia più specifica di  $G_0$ . Dopo che tutti gli esempi sono stati elaborati, lo spazio versione calcolato contiene tutte le ipotesi coerenti con questi esempi.

1. Inizializza  $G$  con l'insieme di ipotesi massimamente generali in  $H$   $G_0 = <?, ?, ?, ?, ?, ? >$
2. Inizializza  $S$  con l'insieme di ipotesi massimamente specifiche in  $H$   $S_0 = <0, 0, 0, 0, 0, 0 >$
3. Per ogni esempio di training  $d = <x, c(x)>$ :
  - se  $d$  è positivo:
    - Rimuovere da  $G$  ogni ipotesi incompatibile con  $d$
    - per ogni ipotesi  $s$  in  $S$  che non è coerente con  $d$ : (Generalizziamo  $S$ )
      - \* Rimuovere  $s$  da  $S$
      - \* Aggiungi a  $S$  tutte le minime generalizzazioni  $h$  di  $s$  tali che  $h$  è consistente con  $d$  e alcuni membri di  $G$  sono più generali di  $h$
      - \* Rimuovi da  $S$  ogni ipotesi più generale di un'altra ipotesi in  $S$
  - se  $d$  è negativo:
    - Rimuovere da  $S$  ogni ipotesi incompatibile con  $d$
    - per ogni ipotesi  $g$  in  $G$  che non è coerente con  $d$ : (Specializziamo  $G$ )
      - \* Rimuovere  $g$  da  $G$
      - \* Aggiungi a  $G$  tutte le minime specializzazioni  $h$  di  $g$  tali che  $h$  è consistente con  $d$  e alcuni membri di  $S$  sono più specifici di  $h$
      - \* Rimuovi da  $G$  ogni ipotesi meno generale di un'altra ipotesi in  $G$

### 3.11.1 Esempio di Candidate Elimination

S:  $\{\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$

G:  $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle +$

S:  $\{\langle \text{Sunny Warm Normal Strong Warm Same} \rangle\}$

G:  $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle +$

S:  $\{\langle \text{Sunny Warm ? Strong Warm Same} \rangle\}$

G:  $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle -$

S:  $\{\langle \text{Sunny Warm ? Strong Warm Same} \rangle\}$

G:  $\{\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle, \langle ?, \text{Warm, ?, ?, ?, ?} \rangle, \langle ?, ?, ?, ?, \text{Same} \rangle\}$

$x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle +$

S:  $\{\langle \text{Sunny Warm ? Strong ? ?} \rangle\}$

G:  $\{\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle, \langle ?, \text{Warm, ?, ?, ?} \rangle\}$

esempio di allenamento negativo, ora specializzo G lascio il più generale dove non collide

G al passaggio 3 contiene le specializzazioni minime che mi permettono di coprire  $x_3$  come negativa, senza rendere G più specializzato di S. S e G sono le "guardie" di tutti i positivi e di tutti i negativi trovati finora. Il fatto che se specializzo troppo G rendo false alcuni input già accettati, deriva dal partial order: intelligentemente il controllo di consistenza di tutti i vecchi input non è effettuato dall'algoritmo. Infatti, il limite S del Version Space costituisce un riepilogo degli esempi positivi riscontrati in precedenza che possono essere utilizzati per determinare se una ipotesi è coerente con questi esempi. Il limite G riassume le informazioni da esempi negativi riscontrati in precedenza. Ogni ipotesi più specifica di G è garantita per essere coerente con esempi negativi del passato.

## 4 Inductive Bias

Il nostro spazio delle ipotesi non è in grado di rappresentare target disgiuntivi come ad esempio  $(Sky = Sunny) \vee (Sky = Cloudy)$ .

Se avessimo due esempi del tipo:

$x_1 = < Sunny, Warm, Normal, Strong, Cool, Change >$  (positivo)

$x_2 = < Cloudy, Warm, Normal, Strong, Cool, Change >$  (positivo)

troveremo che  $S = \{ < ?, Warm, Normal, Strong, Cool, Change > \}$

che è inconsistente con l'esempio  $x_3 = < Rainy, Warm, Normal, Strong, Cool, Change >$  (negativo)

trovando  $S = \{ \}$ , cioè S collassa, perché la funzione target è fatta solo di and.

PROBLEMA: abbiamo vincolato il sistema di apprendimento considerando solo ipotesi congiuntive.

Abbiamo bisogno di uno spazio di ipotesi più espressivo.

### 4.1 Sistema di apprendimento Unbiased

La soluzione ovvia al problema di assicurare che il concetto target sia nello spazio di ipotesi  $H$  è di fornire uno spazio di ipotesi in grado di rappresentare ogni concetto insegnabile.

Significa che  $H$  è l'insieme di tutti i possibili sottoinsiemi di  $X$  (questo insieme viene chiamato anche l'insieme di potenza  $P(X)$ ). In EnjoySport  $|H| = 96$ ,  $|P(X)| = 2^{96}$  concetti distinti. Un'ipotesi può essere rappresentata con disgiunzioni, congiunzioni e negazioni delle nostre precedenti ipotesi.  $H$  sicuramente contiene il concetto target. Con questa modifica quali sono i nostri  $G$  e  $S$ ?

NUOVO PROBLEMA: il nostro algoritmo di apprendimento dei concetti non è ora in grado di generalizzare oltre gli esempi osservati. Assumiamo tre esempi positivi ( $x_1, x_2, x_3$ ) e due esempi negativi ( $x_4, x_5$ ).  $S = \{ (x_1 \vee x_2 \vee x_3) \}$  e  $G = \{ \neg(x_4 \vee x_5) \} \rightarrow$  Non abbiamo generalizzazione,  $S$  sarà sempre la disgiunzione degli esempi positivi e  $G$  la disgiunzione di quelli negativi!

Proprietà: un sistema di apprendimento non vincolato non è in grado di generalizzare

Prova: ogni nuova istanza verrà classificata positivamente precisamente da metà delle ipotesi nel Version Space e negativa dell'altra metà. Poiché  $H$  è l'insieme di potenza di  $X$  e sia  $x_i$  una istanza non vista precedentemente,  $\forall h$  consistente con  $x_i(test)$ ,  $\exists h'$  identico ad  $h$  con eccezione che  $h'(x_i) \neq h(x_i)$  cioè  $h \in VS \rightarrow h' \in VS$  (sono identici in D)

#### 4.1.1 Il learning Unbiased è utile?

Un sistema di apprendimento che non fa assunzioni preliminari riguardo all'identità del concetto target non ha basi razionali per classificare eventuali istanze non conosciute. Un sistema di apprendimento dovrebbe essere in grado di generalizzare a partire dai dati di allenamento al fine di classificare istanze mai viste precedentemente. Il "bias" non è solo assunto per efficienza, ma è necessario per la generalizzazione, tuttavia non ci dice quale sia la migliore soluzione per la generalizzazione.

## 4.2 Definizione Formale di Inductive Bias

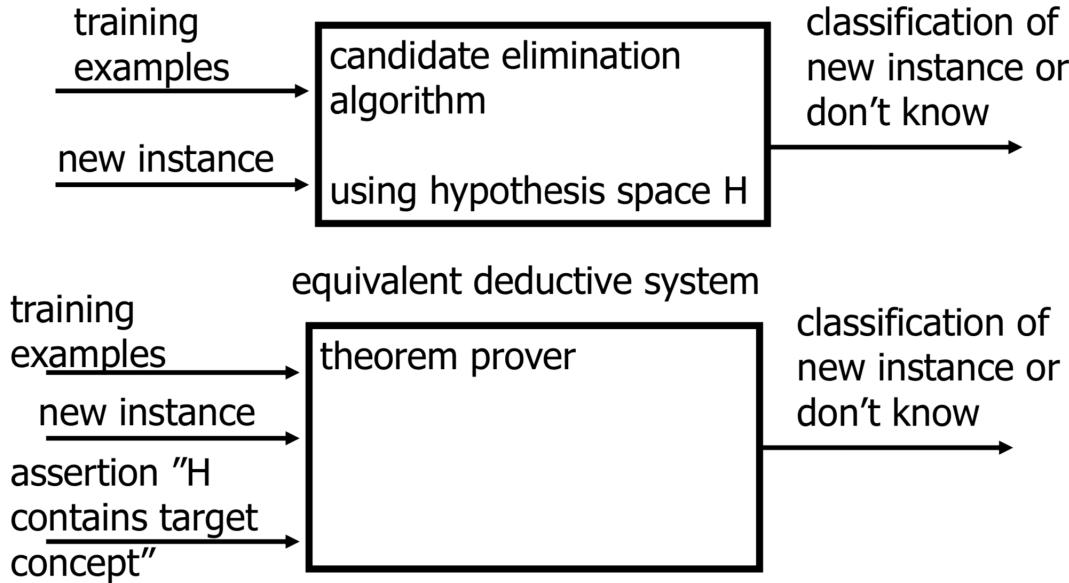
Consideriamo:

- Un algoritmo di apprendimento concettuale  $L$
- Un insieme di istanze  $X$  e un concetto target  $c$
- Sia  $D_c = \{ < x, c(x) > \}$  un insieme di esempi di training di  $c$
- Sia  $L(x_i, D_c)$  la classificazione assegnata all'istanza  $x_i$  da  $L$  dopo l'allenamento su  $D_c$ .

L'Inductive Bias di  $L$  è un insieme minimo di asserzioni  $B$  tale che per qualsiasi concetto target  $c$  e corrispondenti dati di allenamento  $D_c$ :

$(\forall x_i \in X)[B \wedge D_c \wedge x_i] \vdash L(x_i, D_c)$  dove  $A \vdash B$  significa che  $B$  è deducibile da  $A$ .

### 4.3 Sistemi Induttivi e Deduttivi equivalenti



Il comportamento input-output dell'algoritmo Candidate Elimination usando uno spazio delle ipotesi  $H$  è lo stesso di un sistema deduttivo in cui diamo come input l'asserzione " $H$  contiene il concetto target". Questa asserzione è chiamata Bias Induttivo dell'algoritmo.

Il Bias è l'insieme delle asserzioni che ci permette di trasformare il problema da induttivo a deduttivo.

### 4.4 Tre (algoritmi) sistemi di apprendimento con Bias differenti

1. Rote Learner: (lookup table) L'apprendimento corrisponde semplicemente alla memorizzazione di ogni esempio di allenamento osservato nella memoria. Le istanze successive vengono classificate osservandole nella memoria. Se l'istanza viene trovata in memoria, viene restituita la classificazione memorizzata. In caso contrario, il sistema rifiuta di classificare la nuova istanza. In poche parole salva gli esempi, classifica  $x$  se e solo se si "accoppiano" con uno degli esempi visti precedentemente.  
Non c'è Bias Induttivo → nessuna generalizzazione
2. Candidate Elimination: Le nuove istanze vengono classificate solo nel caso in cui tutti i membri del Version Space corrente concordino la classificazione. In caso contrario, il sistema rifiuta di classificare la nuova istanza.  
Bias Induttivo → il concetto target può essere rappresentato nel suo spazio di ipotesi.
3. Find-S: Questo algoritmo, descritto in precedenza, trova l'ipotesi più specifica coerente con gli esempi di addestramento. Quindi utilizza questa ipotesi per classificare tutte le istanze successive.  
Bias Induttivo → il concetto target può essere rappresentato nel suo spazio delle ipotesi e tutte le istanze sono istanze negative a meno che l'opposto non sia implicato da altre sue conoscenze.

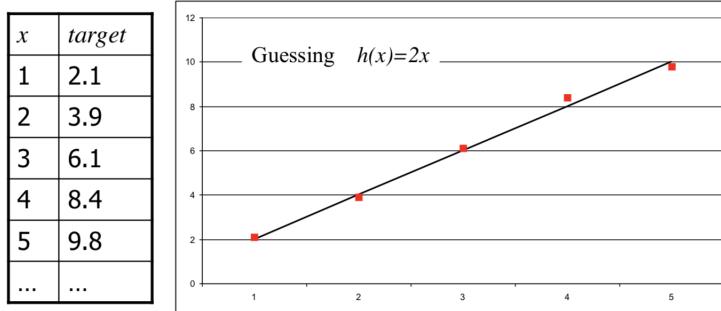
## 5 Modelli Lineari

Abbiamo visto che lo spazio delle ipotesi  $H$  costituisce l'insieme delle funzioni che possono essere realizzate dal sistema di apprendimento. Si assume che la funzione da apprendere  $f$  possa essere rappresentata da una ipotesi  $h$  in  $H$  (selezione di  $h$  attraverso i dati di apprendimento) o che almeno una ipotesi  $h$  in  $H$  sia simile a  $f$  (approssimazione). Abbiamo compreso che un Algoritmo di Ricerca nello Spazio delle Ipotesi è rappresentato in ML tramite un algoritmo di apprendimento (ad esempio adattamento dei parametri liberi del modello al task). NOTA:  $H$  non può coincidere con l'insieme di tutte le funzioni possibili ma la ricerca deve essere esaustiva → Bias Induttivo.

Sia la regressione che la classificazione (quest'ultima vista nel Concept Learning) appartengono alla categoria delle tecniche di Supervised Learning, ovvero un tipo di apprendimento che fornisce a priori una serie di dati e informazioni al sistema, prima che questo cominci ad apprendere. Andremo a vedere come i Modelli lineare possono essere usati sia per la regressione (Regressione lineare) sia per la classificazione. La regressione lineare si differenzia nettamente dalla classificazione, poiché la classificazione si limita a discriminare gli elementi in un determinato numero di classi, mentre nella regressione l'input è un dato e il sistema ci restituisce un output reale approssimando una funzione. La regressione è un processo statistico che cerca di stabilire una relazione tra due o più variabili. Fornendo a un modello di regressione un valore  $x$ , questo restituirà il corrispondente valore  $y$  generato dall'elaborazione di  $x$ .

### 5.1 Esempio di regressione

La regressione come abbiamo detto è un processo di stima di una funzione a valori reali sulla base di una serie finita di campioni "rumorosi" (conosciamo le coppie  $(x, f(x)+\text{random noise})$ ).

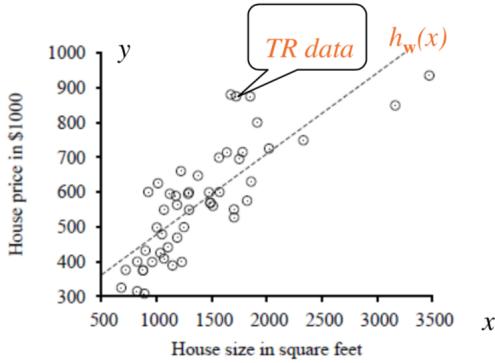


### 5.2 Modello di Regressione Lineare Semplice (Univariata)

Si parte con una variabile input  $x$  e una in output  $y$ , assumiamo il modello  $h_w(x)$  espresso come  $y = w_1x + w_0$  dove  $w_0$  e  $w_1$  sono parametri liberi a valori reali da apprendere. Usiamo la lettera  $w$  perché immaginiamo questi coefficienti come weights (pesi) poiché il valore  $y$  cambia in base a questi valori. Come si può intuire cerchiamo di adattarci ai dati con una linea retta.

Lavoriamo in uno spazio di ipotesi infinite (i valori  $w_i$  sono continui) ma abbiamo una buona soluzione grazie alla matematica classica. Gauss dimostrò che se i valori  $y$  hanno del "rumore" distribuito allora i valori  $w_1$  e  $w_2$  possono essere trovati minimizzando la somma dei quadrati degli errori (dopo vedremo meglio cosa significa).

Assumiamo che la variabile  $y$  sia (linearmente) correlata a un'altra variabile  $x$  o ad altre variabili per cui  $y = w_1x + w_0 + \text{noise}$ , dove i  $w_i$  sono i parametri liberi. Il "noise" è l'errore nella misurazione dei valori target con distribuzione normale. Cerchiamo di costruire un modello (trovare le variabili  $w_i$ ) per predire/stimare la  $y$  per dei valori  $x$  non visti precedentemente.



Un esempio di regressione lineare per il problema sovrastante (bisogna associare il prezzo delle case in base alla metratura.  $y$  costo della casa,  $x$  metratura) è la funzione  $h_w(x) = 0.232 * x + 246$  ma come arrivare a questa funzione?

### 5.3 Costruzione della funzione via LMS

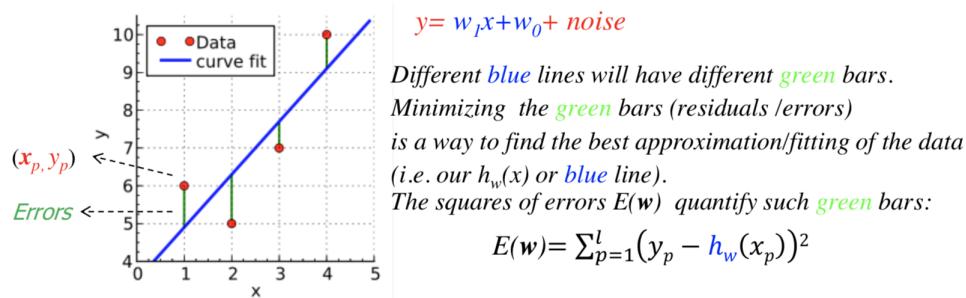
L'algoritmo LeastMeanSquare consiste nel trovare i  $w_i$  per cui l'errore è minimizzato (miglior adattamento dei dati sul training set con  $l$  esempi).

- Dato un insieme di  $l$  esempi di allenamento salvati come coppie  $(x_p, y_p)$
- Trovare  $h_w(x) = w_1x + w_0$  che minimizza l'errore medio sul Training Set
- Per calcolare la funzione di errore Loss usiamo la somma dei quadrati delle differenze tra il valore dato dall'esempio  $y_p$  e il valore calcolato dalla funzione  $h_w(x)$ . Il quadrato serve per avere solo valori positivi (si potrebbe anche usare il valore assoluto, ma è meglio di no, e il perché lo vedremo in seguito)

$$Loss(h_w) = E(w) = \sum_{p=1}^l (y_p - h_w(x_p))^2 = \sum_{p=1}^l (y_p - (w_1 * x_p + w_0))^2 \quad (1)$$

dove  $x_p$  è il p-esimo input e  $y_p$  è il p-esimo output dell'esempio p. Dividendo per l ottengo la media.

Quindi dobbiamo trovare l'argomento minimo  $w$  tale che l'errore è minimo in L2 (norma 2, le norme inducono una distanza):  $w = argmin_w Loss(h_w) = argmin_w E(w)$  in L2. A livello grafico lo rappresentiamo così:



Il metodo dei minimi quadrati è un approccio standard alla soluzione approssimata di sistemi sovr-determinati, ovvero insiemi di equazioni in cui vi sono più equazioni che incognite.

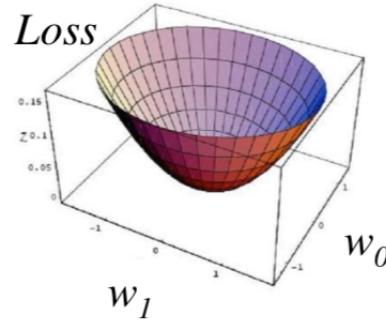
Il minimo locale é un punto stazionario dove il gradiente é nullo.

$$\frac{\partial E(w)}{\partial w_i} = 0 \quad \text{per } i = 0 \dots \text{dimensione\_spazio} \quad (2)$$

per la regressione lineare semplice abbiamo che la funzione di Loss é minimizzata quando le sue derivate parziali rispetto a  $w_0$  e  $w_1$  sono 0:

$$\frac{\partial E(x)}{\partial w_0} = 0 \quad \frac{\partial E(x)}{\partial w_1} = 0 \quad (3)$$

Se la funzione di perdita é convessa abbiamo la seguente soluzione diretta poiché non abbiamo nessun minimo locale.



per  $w_1$  abbiamo che

$$w_1 = \frac{l * (\sum x_p y_p) - (\sum x_p)(\sum y_p)}{l * (\sum x_p^2) - (\sum x_p)^2} \quad (4)$$

mentre per  $w_0$

$$w_0 = \frac{\sum y_p - w_1 * \sum x_p}{l} \quad (5)$$

Un approccio differente serve invece per quelle equazioni per cui il minimo della funzione di Loss spesso non ha una soluzione definita: infatti useremo una sorta di Hill Climbing iterativo che segue il gradiente discendente. Sfruttiamo le seguenti regole delle derivate parziali:

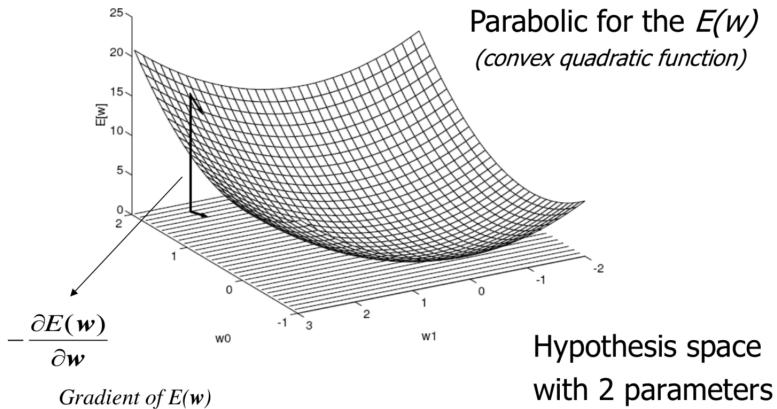
$$\frac{\partial}{\partial w} k = 0 \quad \frac{\partial}{\partial w} w = 1 \quad \frac{\partial}{\partial w^2} w = 2w \quad \frac{\partial(f(w))^2}{\partial w} = 2f(w) \frac{\partial f(w)}{\partial w} \quad (6)$$

ora cerchiamo il gradiente

$$\frac{\partial E(w)}{\partial w_i} = \frac{\partial}{\partial w_i} (y - h_w(x))^2 = 2(y - h_w(x)) \frac{\partial}{\partial w_i} (y - h_w(x)) = 2(y - h_w(x)) \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)) \quad (7)$$

applicando  $w_0$  e  $w_1$  nella derivata ottengo due equazioni

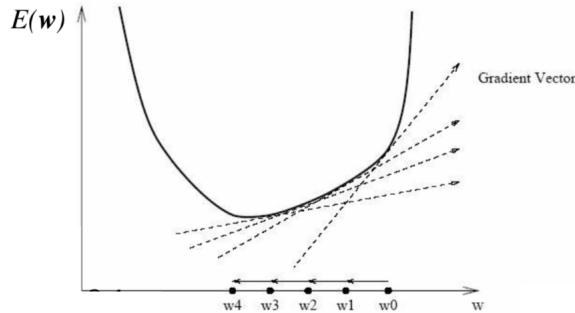
$$\frac{\partial E(w)}{\partial w_0} = -2(y - h_w(x)) \quad \text{e} \quad \frac{\partial E(w)}{\partial w_1} = -2(y - h_w(x)) * x \quad (8)$$



### 5.3.1 Gradiente discendente

La derivazione precedente suggerisce la costruzione di un algoritmo iterativo basato su  $\partial E(w)/\partial w_i$ . Il gradiente ci indica la direzione di salita, possiamo spostarci verso il minimo con discesa del gradiente  $\Delta w = -\text{gradiente}E(w)$ . La ricerca locale inizia con il vettore che contiene i valori dei pesi, viene modificato iterativamente per diminuire fino a minimizzare l'errore della funzione.

$$w_{new} = w_{old} + \eta * \Delta w \text{ dove } \eta \text{ é una costante che indicare il rateo di apprendimento}$$



Le nostre regole di correzione dell'errore (chiamate delta-rule) cambiano i valori w proporzionalmente all'errore:

$$\Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2(y - h_w(x)) \quad e \quad \Delta w_1 = -\frac{\partial E(w)}{\partial w_1} = 2(y - h_w(x)) * x \quad (9)$$

- se  $(y_{target} - output) = 0$  significa che non abbiamo errore e quindi nessuna correzione da fare
- se  $(output > y_{target})$  cioè  $(y - h) < 0$  vuol dire che l'output trovato é troppo alto quindi:
  - $\Delta w_0$  negativo → riduco  $w_0$
  - se  $(x_{input} > 0)$  e  $\Delta w_1$  negativo → riduco  $w_1$  altrimenti incremento  $w_1$
- se  $(output < y_{target})$  cioè  $(y - h) > 0$  vuol dire che l'output trovato é troppo basso quindi:
  - $\Delta w_0$  negativo → aumento  $w_0$
  - se  $(x_{input} < 0)$  e  $\Delta w_1$  positivo → aumento  $w_1$  altrimenti diminuisco  $w_1$

Questo ci permette la ricerca in uno spazio delle ipotesi infinito, e può essere applicato facilmente per spazi  $H$  continui e con funzione di Loss differenziabile (altrimenti non posso calcolare il gradiente!). Per questo motivo il valore assoluto è scomodo: potrebbe rendere non differenziabile la funzione. Arriviamo quindi a definire le funzioni su  $l$  esempi:

$$\Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2 \sum_{p=0}^l (y_p - h_w(x_p)) \quad e \quad \Delta w_1 = -\frac{\partial E(w)}{\partial w_1} = 2 \sum_{p=0}^l (y_p - h_w(x_p)) * x_p \quad (10)$$

Ora può venirci in mente una domanda, aggiorniamo i  $w_i$  dopo aver ispezionato un insieme di esempi di allenamento 1, oppure dopo ogni singolo esempio analizzato?

- Batch Algorithm: calcoliamo la sommatoria dell'errore quadratico su tutti i pattern diversi e otteniamo il gradiente, dopo i dati di training sommati abbiamo una “epoca” e aggiorniamo i  $w_i$ . Può essere molto lento...
- On-Line Algorithm: calcoliamo il gradiente su un pattern  $p$  e poi aggiorniamo subito  $w_i$ . Solitamente è più veloce del Batch...

## 5.4 Modello di Regressione Lineare Multivariato

Il caso standard è con due variabili, ma in realtà possiamo avere in input centinaia di variabili, l'esempio sul prezzo delle case in realtà conteneva anche le variabili che indicavano il numero delle stanze, l'età della casa ecc...

### 5.4.1 Notazione dei dati

Pattern	$x_1$	$x_2$	$x_j$	$x_n$
Pat 1	$x_{1,1}$	$x_{1,2}$		$x_{1,n}$
...				
Pat $p$	$x_{p,1}$	$x_{p,2}$	$x_{p,j}$	$x_{p,n}$
...				

$X$  è una matrice  $l$  righe \*  $n$  colonne (numero di pattern \* numero di variabili), ogni riga della matrice  $X_i$  è un vettore che rappresenta un esempio.  $x_{i,j}$  è uno scalare cioè la componente  $j$  della riga  $i$ . Stiamo quindi dicendo che ogni esempio  $x_p$  è un vettore di  $n$  elementi.

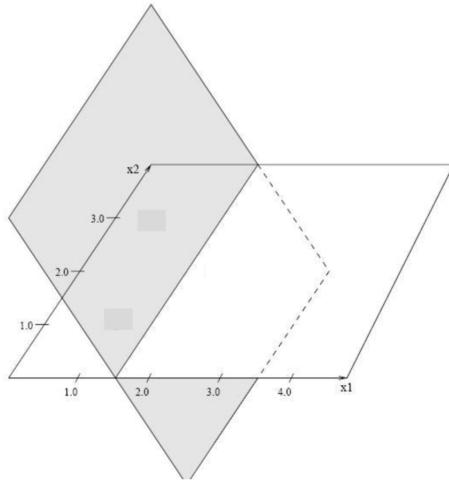
Supponendo vettori colonna  $x$  e  $w$ , con numero di esempi  $l$  e dimensione in input  $n$ , possiamo scrivere

$$w^T x + w_0 = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n \quad (11)$$

dove  $w_0$  è il bias (non c'entra con il Concept Learning) è la distanza dalla componente 0. Possiamo anche scrivere che  $x_0 = 1$  in modo tale da poter scrivere che  $w^T x = x^T w$  quindi:  $x^T = [1, x_1, \dots, x_n]$  e  $w^T = [w_0, w_1, \dots, w_n]$

## 5.5 Hyperplane

Anziché adattare una retta, adattiamo un iperpiano.

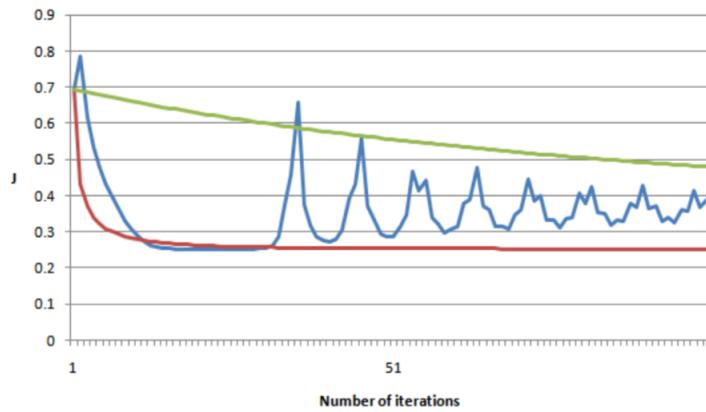


quindi abbiamo che  $w^T x = w_1 x_1 + w_2 x_2 + w_0$  e  $h(x_p) = x_p^T w = \sum_{i=0}^n x_{p,i} w_i$  dove  $x_{p,i}$  è la componente i-esima del p-esimo esempio.

## 5.6 Gradiente Discendente Algoritmo

1. si inizia con il vettore dei pesi  $w_{iniziale}$  e si fissa un  $0 < \eta < 1$
2. si calcola il gradiente  $\Delta w = -\text{gradiente}E(w) = -(\partial E(w)/\partial w)$
3. si calcola  $w_{new} = w_{old} + \eta * \Delta w$
4. ripetere dal passo 2 finché  $E(w)$  è sufficientemente piccolo

La versione batch dell'algoritmo si occupa di calcolare  $\Delta w$  dopo un insieme di esempi l. La versione online invece aggiorna  $\Delta w$  dopo ogni esempio p al posto di aspettare la sommatoria su l, ma ha bisogno di uno step size  $\eta$  più piccolo.



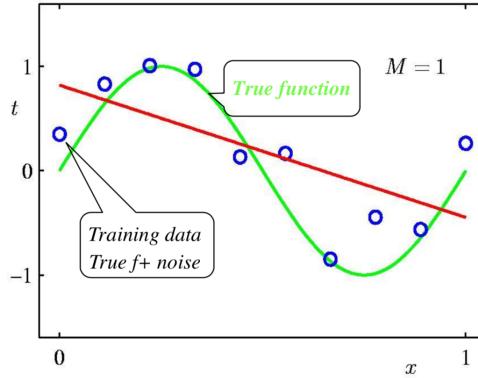
Questi sono tre esempi di curve, dove noi andremo a preferire quella che diminuisce nel minor tempo possibile (quella che impiega meno iterazioni).

## 5.7 Vantaggi dei modelli lineari

Se funziona bene è un modello "meraviglioso", molto semplice dove tutte le informazioni sui dati sono contenute nel vettore  $w$ . Facile da interpretare tant'è che viene usato in medicina, biologia, chimica, economia... Una delle cose fondamentali è che sono ammessi dati rumorosi e viene usato/incluso in modelli più complessi.

## 5.8 Limitazioni

La regressione lineare presenta delle limitazioni quando si parla di problemi non lineari.



Notiamo che  $h_w(X) = w_1x + w_0$  definita "lineare" non significa che é una linea retta, ma piuttosto come il modo in cui i coefficienti  $w$  occorrono nell'equazione di regressione (cioè lineare in  $w$ , non in  $x$ ). Quindi, possiamo anche trasformare gli input, come  $x, x^2, x^3, \dots$  con relazioni non lineari tra input e output. Sfruttando l'algoritmo LMS utilizzato fino ad ora:

$$y(x, w) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j \quad (12)$$

### 5.8.1 Esempio

Rispetto all'equazione scritta precedente in questo caso abbiamo  $M=2$ . La funzione non deve essere lineare nell'argomento (variabili in input) ma solo nei parametri che sono determinati per dare il miglior adattamento.

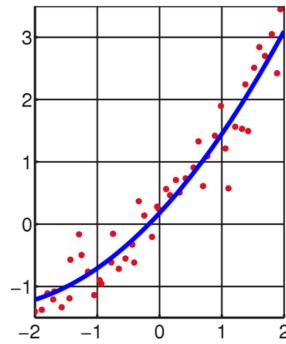


Figure 3: fitting su una funzione quadratica (linea blu) attraverso un set di dati (punti rossi)

## 5.9 Linear Basis Expansion

Prima calcolavamo il prodotto scalare tra  $w$  e  $x$  ( $w$  trasposto per  $x$ ) ora prendo delle trasformazioni (anche non lineari) del vettore  $x$  e le uso come un'espansione della base, combinandole in maniera lineare in accordo con la funzione  $\phi_k$ .

$$h_w(x) = \sum_{k=0}^K w_k \phi_k(x) \quad (13)$$

La funzione  $\phi$  può essere espressa come rappresentazioni polinomiali del tipo  $\phi(x) = x_j^2$  oppure  $\phi(x) = x_j * x_i$ , o ancora trasformazioni non lineari come  $\phi(x) = \log(x_j)$ . Il modello è rimasto lineare nei parametri (è sempre  $x$ !) quindi possiamo usare lo stesso algoritmo di prima.

### 5.9.1 Esempi

- dimensione di  $x = 1$  e  $\phi_j(x) = x^j$

$$h(x) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j \quad (14)$$

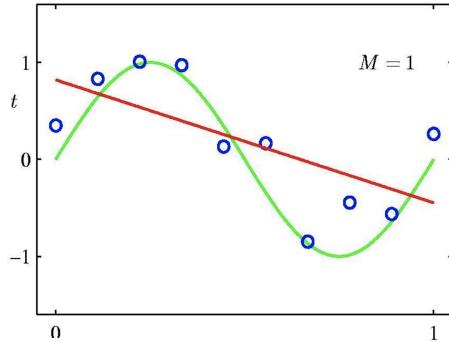
regressione polinomiale ad 1 dimensione ( $K=M$ )

- $\phi(x) = \phi([x_1, x_2])$

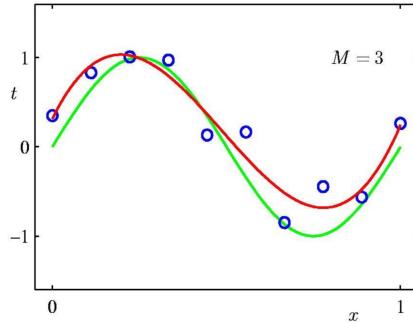
$$h(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 \quad (15)$$

Quale  $\phi$  scegliere? Adottiamo un modello in stile dizionario? I punti a favore sono la maggiore espressività poiché può modellare relazioni più complesse di quelle lineari. I punti contro sono l'avere una grossa base di funzioni che richiedono metodi per controllare la complessità del modello, vediamo perché...

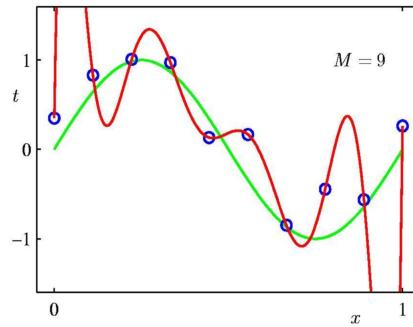
- grado del polinomio = 1, notiamo che è esegue troppo "underfitting".



- ordine del polinomio = 3, va bene ma potrebbe servirci più flessibilità.



- grado del polinomio = 9, notiamo che è flessibilissimo ma eccessivo. Cioè  $E(w)=0$  ma dobbiamo tenere conto degli errori sul test set, perché se i dati fossero rumorosi noi stiamo adattando al rumore.



### 5.9.2 Tradeoff per la complessità

- Se scegliamo un modello troppo semplice non fitta bene i dati e abbiamo una soluzione vincolata (ce ne accorgiamo perché  $E(w)$  è alto) → Underfitting
- Se scegliamo un modello troppo complesso siamo troppo sensibili alla piccole perturbazioni dei dati → Overfitting

Vogliamo scegliere la regolarizzazione per bilanciare bias e varianza e lo faremo attraverso il controllo della complessità del modello.

## 5.10 Regolarizzazione - Ridge Regression

Possiamo evitare l'overfitting penalizzando le funzioni complesse mantenendo comunque la flessibilità dello spazio delle ipotesi (un po' come il rasoio di Occam, si preferisce lo spazio delle ipotesi più semplice che fitta i dati).

La Ridge Regression, chiamata anche regolarizzazione di Tikhonov, è sempre un modello LMS ma regolarizzato. Per esempio è possibile aggiungere vincoli alla sommatoria del valore di  $|w_j|$  favorendo modelli "sparsi" (sono modelli con meno termini dovuti a pesi  $w_j = 0$ ).

Un modello non regolarizzato fitta bene ma generalizza male (non ci serve a niente). Vorrei un modello semplice (magari con alcuni coefficienti che valgono zero). Come si fa? Aggiungiamo una penalizzazione alla complessità del modello della loss, sommando la norma del vettore  $w$ , moltiplicato per un parametro  $\lambda$  chiamato coefficiente di regolarizzazione. Con il parametro lambda (che è una costante), non ho più bisogno di modificare il grado dei polinomi, o analizzare funzioni più complesse.

$$\text{definito } E_D(w) + \lambda E_W(w) \text{ otteniamo } \text{Loss}(h_w) = \sum_p (y_p - h_w(x_p))^2 + \lambda \|w\|^2 \quad (16)$$

Ottenendo quindi la nuova regola di apprendimento caratterizzata dal weight decay (aggiunta di  $2\lambda w$ )

$$w_{new} = w_{old} + \eta * \Delta w - 2\lambda w_{old} \quad (17)$$

Per via di  $-2\lambda w_{old}$  decremento il parametro se positivo, lo incremento se negativo (in pratica lo avvicino a zero  $\rightarrow$  weight decay). Quindi possiamo controllare la complessità del polinomio sfruttando solamente  $\lambda$ .

Finora abbiamo posto tutta la complessità nel bias di linguaggio, cioè un vincolo posto sul modello (lineare, quadratico, etc). Con la regolarizzazione spostato tutto sul bias di ricerca, cioè sull'algoritmo, come l'algoritmo riesce a trovare la soluzione. Noi non poniamo a priori una complessità al modello, ma sarà il mio metodo di ricerca intelligente a scegliere la funzione più semplice.

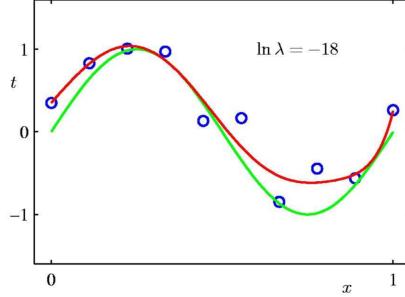


Figure 4: Polinomio di grado 9 con  $\log_e(\lambda) = -18$  quindi  $\lambda$  basso

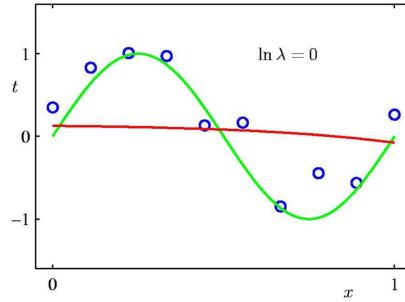


Figure 5: Polinomio di grado 9 con  $\log_e(\lambda) = 0$  quindi  $\lambda$  troppo alto

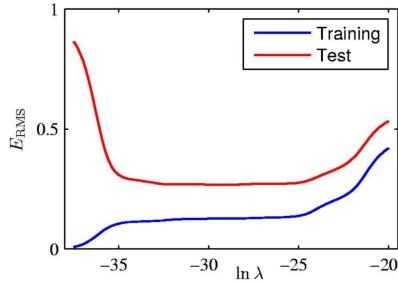


Figure 6:  $E_{RMS}$  vs  $\log_e(\lambda)$

Possiamo concludere quindi dicendo che se  $\lambda$  è basso  $\rightarrow$  overfitting, mentre se  $\lambda$  alto  $\rightarrow$  underfitting.

### 5.11 Limitazione delle funzioni a base fissa

Quando facciamo un'espansione polinomiale, la dimensionalità del problema cresce molto, e i dati che prima erano densi, potrebbero diventare sparsi. Quando la dimensione dell'input aumenta, il problema diventa enormemente difficile (la difficoltà cresce esponenzialmente, proprio come il volume con le dimensioni). Inoltre, se inizio a mettere dati a caso, rischio di aumentare il rumore. Avere la funzione base che opera su ogni dimensione di uno spazio implica che lo spazio di input richieda un numero combinatorio di funzioni. Per esempio un polinomio dell'ordine 3, utilizza tutte le combinazioni di variabili di input dovute ai prodotti  $x_1x_2, x_2x_3, \dots, x_1x_2x_3$  ecc...  $\rightarrow D^3$  (approssimare all'aumentare di D)

In altri modelli, vedremo come possiamo farcela con meno funzioni di base. Si scelgono le funzioni dal dizionario e si mettono dei parametri liberi, per adattare la  $\phi$  ad H. Ho il vantaggio di essere più generale, ma il modello diventa non lineare rispetto a w (vedremo nelle reti neurali).

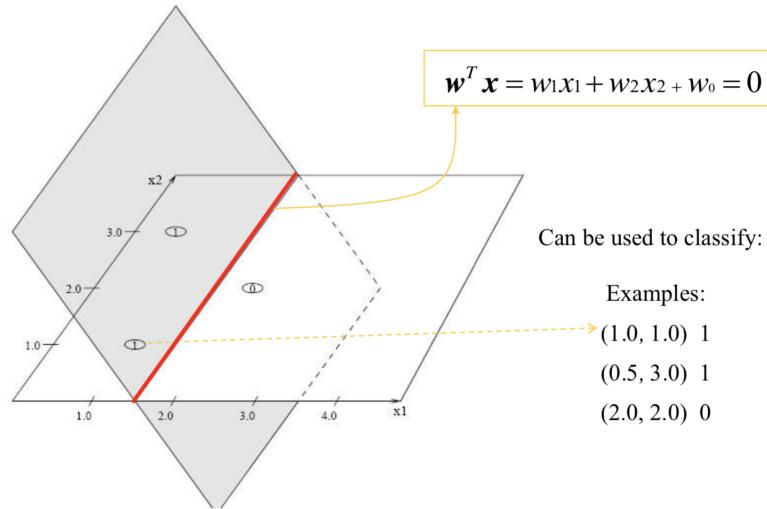
In altri modelli invece di calcolare esplicitamente la  $\phi$ , viene fatta in maniera implicita, col concetto di funzioni kernel, controllando la complessità del modello direttamente con la funzione obiettivo, non più con una penalità aggiunta dopo (SVM).

## 5.12 Classificazione con il modello lineare

Possiamo utilizzare il modello lineare per la classificazione, in questo caso usiamo un iperpiano ( $w^T * x$ ) assumendo valori positivi o negativi. Sfruttiamo questi modelli per decidere se un punto  $x$  appartiene alla zona positiva o negativa dell'iperpiano. Quindi vogliamo impostare  $w$  vettore in modo tale da ottenere una buona precisione di classificazione.

### 5.12.1 Visione geometrica dell'iperpiano

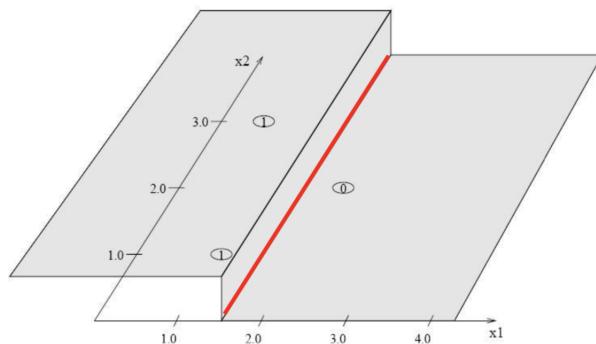
$w^T * x$  (prodotto di vettore riga per vettore colonna) definisce un iperpiano, il Decision Boundary è il luogo geometrico<sup>3</sup> dei punti dove l'iperpiano vale zero e può essere utilizzato per classificare.



Quindi possiamo definire una funzione di classificazione del tipo:

$$h(x) = \begin{cases} 1 & \text{if } w^T x + w_0 \geq 0. \\ 0 & \text{altrimenti.} \end{cases} \quad (18)$$

che possiamo scrivere come  $h(x) = \text{segno}(w^T x + w_0) \rightarrow h(x_p) = \text{sign}(x_p^T w) = \text{sign}(\sum_{i=0}^n x_{p,i} w_i)$



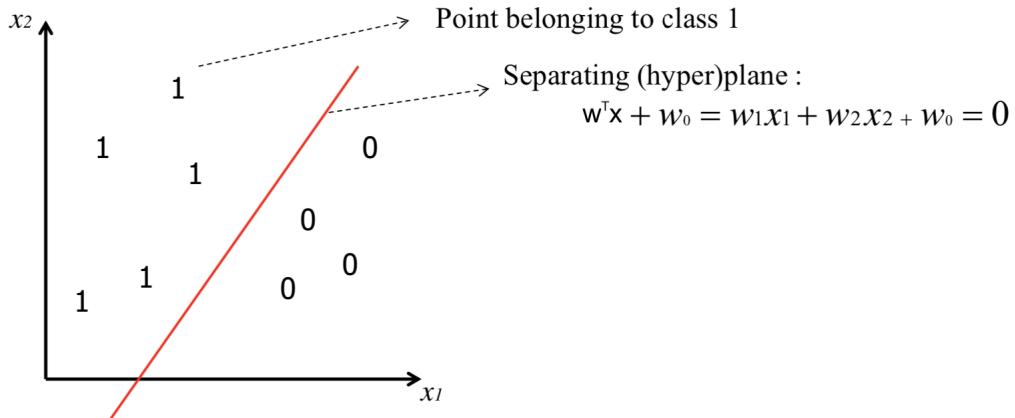

---

<sup>3</sup>è una linea, o un piano (quando ci troviamo in dimensioni più gradi)

### 5.12.2 Classificazione attraverso Decision Boundary lineare

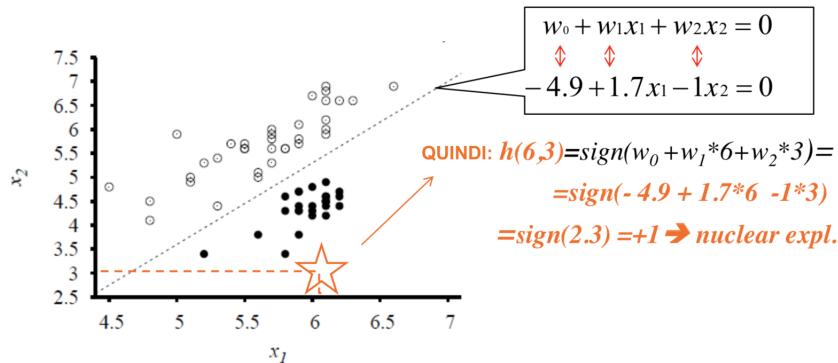
La classificazione può essere vista come l'allocazione dello spazio in input in diverse regioni di decisione. Ad esempio in uno spazio di due dimensioni  $x = (x_1, x_2)$  in  $R^2$  e con  $f(x) = 0 \vee 1$  il decision boundary sarà una retta.

Linear Threshold Unit (LTU) = indicatore di funzioni, quanti ne abbiamo? un insieme di divisioni indotte da iperpiani, sono tutti i modi che abbiamo per muovere la linea rossa.



### 5.12.3 Esempio Terremoti/Esplosioni Nucleari

Trovare  $h$  in modo tale che il dato  $(x_1, x_2)$  ritorni  $(0 \vee -1)$  per i terremoti e  $(1)$  per esplosioni nucleari. I dati sismici sono rappresentati da  $x_1$  che è la magnitudine dell'onda del corpo e  $x_2$  che è la magnitudine dell'onda della superficie. L'algoritmo trova il decision boundary.



### 5.12.4 Esempio Email Spam

Trovare  $h(\text{mail})$  in modo tale che restituisca  $+1$  se è spam e  $-1$  se non è una mail spam.

Ad esempio  $\phi(\text{mail}) = \text{txtmail.contains("denaro gratis")}$ ,  $w$  vettore di pesi per le funzioni di input che aiutano alla previsione, cioè peso positivo per "denaro gratis", negativo altrimenti.  $w^T x$  è la combinazione dei pesi.  $h_w(x)$  provvede a dare il limite per decidere se una mail è spam o non è spam.

$$h_w(x) = \text{segno}(\sum_k w_k \phi_k(x)) \quad (19)$$

## 5.13 Il problema di apprendimento (per classificatori lineari)

Assumendo che usiamo ancora il metodo dei minimi quadrati, dato un insieme grande  $l$  di esempi di allenamento, bisogna trovare  $w$  che minimizza la somma residua dei quadrati

$$E(w) = \sum_{p=0}^l (y_p - x_p^T w)^2 = \|y - x^T w\|^2 \quad (20)$$

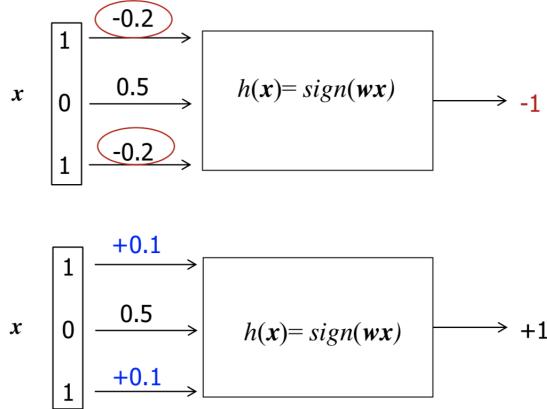
se  $y_p = 1$  allora  $x_p^T w \rightarrow 1$  mentre se  $y_p = 0 \vee -1$  allora  $x_p^T w \rightarrow 0 \vee -1$ .

Possiamo utilizzare l'algoritmo di discesa del gradiente:

$$\Delta w_i = -\frac{\partial E(w)}{\partial w_i} = \sum_{p=1}^l (y_p - (x_p^T w)) * x_{p,i} \quad (21)$$

Nella derivata ignoriamo il 2 che viene fuori dal quadrato. È una costante che non incide sull'eta (il coefficiente di apprendimento).

### 5.13.1 $\Delta w$ come regola della correzione dell'errore



Se un input viene classificato erroneamente dalla funzione  $h(x) = \text{segno}(w^T x)$ , quindi ad esempio il  $\Delta$  negativo per  $w_1$  e  $w_3$ , bisogna aumentare  $w_1$  e  $w_3$  proporzionalmente (con il coefficiente eta). Eseguiamo una correzione d'errore alzando i valori dove sono troppo bassi, e viceversa.

## 5.14 Classificazione dei pattern

Il modello viene *allenato* su un insieme di esempi con il calcolo LMS su  $w^T x$  dall'algoritmo di discesa del gradiente, usato per la regressione lineare. Il modello viene *utilizzato* per la classificazione applicando la funzione di soglia  $h(x) = \text{segno}(w^T x)$ . L'errore può essere calcolato come errore di classificazione o numero di pattern classificati erroneamente.

$$L(h(x_p), d_p) = \begin{cases} 0 & \text{se } h(x_p) = d_p \\ 1 & \text{altrimenti.} \end{cases} \quad \text{mean\_err} = \frac{1}{l} \sum_{i=1}^l L(h(x_i), d_i) \quad (22)$$

dove  $d_p$  è l'output dato dagli esempi di allenamento, quindi stiamo classificando i valori calcolati dalla nostra  $h$  con i valori degli esempi etichettati.

Accuratezza = media dei pattern correttamente classificati =  $(l - numerr)/l$

dove  $numerr = \sum_{i=1}^l L(h(x_i), d_i)$

L'algoritmo è lo stesso come per la regressione → Algoritmo gradiente discendente (anche l'espansione lineare e Tikhonov possono essere applicati). Notare che l'algoritmo converge asintoticamente al MinLeastSquare, non necessariamente al numero minimo di errori di classificazione.

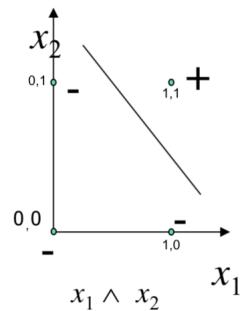
## 5.15 Congiunzioni nel caso di Modello Lineare

Possiamo rappresentare congiunzioni con i modelli lineari, ad esempio:

$x_1 \wedge x_2 \wedge x_4 \Leftrightarrow y$  cioè  $1x_1 + 1x_2 + 0x_3 + 1x_4 > 2$  dove  $w_1 = 1, w_2 = 1, w_3 = 0, w_4 = 1, w_0 = 2$  (vettore dei pesi)

Altro esempio con grafico:

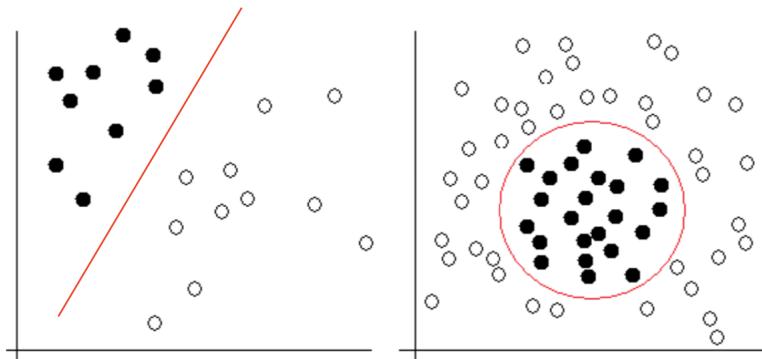
$x_1 \wedge x_2$  cioè  $1x_1 + 1x_2 > 1$



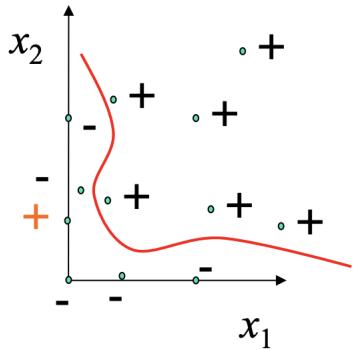
In generale w può essere appreso per trovare la soluzione.

### 5.15.1 Limitazioni

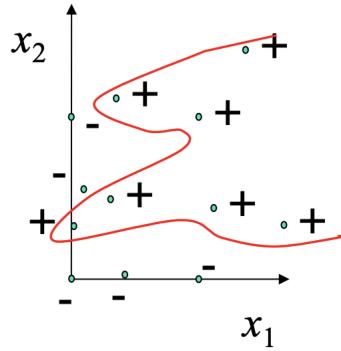
Nella geometria due insiemi di punti in un grafico a due dimensioni sono linearmente separabili quando possono esser separati da una singola linea.



In generale due gruppi di punti sono linearmente separabili in uno spazio di dimensione  $n$  se possono essere separati da un iperpiano di dimensione  $n - 1$ . Il confine lineare dà la soluzione esatta solo per insiemi di punti linearmente separabili.



**Non linear decision boundary  
(by a basis expansion)**  
In this example 1 error is admitted  
(orange)



**A not-smooth classifier  
that may need  
regularization**

## 5.16 Altri sistemi di apprendimento per la classificazione

- Linear Discriminant Analysis
- Logistic Regression: Stima la/le proprietà di  $x$  dato  $y$ ,  $P(y/x)$  partendo modellando la densità della classe come densità nota. Abbiamo una soglia soft (continua, differenziabile) con funzione logica (anziché soglia hard 0/1)
- Reti Neurali e SVM: modelli flessibili che includono approssimazione non lineare sia per la regressione che per la classificazione.

## 5.17 Conclusioni sui Modelli Lineari

I modelli lineari sono un approccio fondato di base per la regressione e la classificazione, è un modo molto compatto per rappresentare la conoscenza (tutti i dati sono nel vettore  $w$ ) ma con una forte assunzione sulla relazione tra i dati (Assumiamo che ci sia una relazione lineare tra i dati, o comunque in accordo alla  $\phi$  che abbiamo scelto. È una assunzione molto forte sullo spazio delle ipotesi). Abbiamo visto un algoritmo di correzione iterativa dell'errore (LMS) che ricerca spazi di ipotesi continui, una visione dei limiti ad approcci lineari e delle esigenze per modelli di ML più flessibili e anche i loro problemi: un'estensione del modello lineare per attività non lineari e un'introduzione al controllo della complessità (regolarizzazione).

## 6 Alberi di decisione

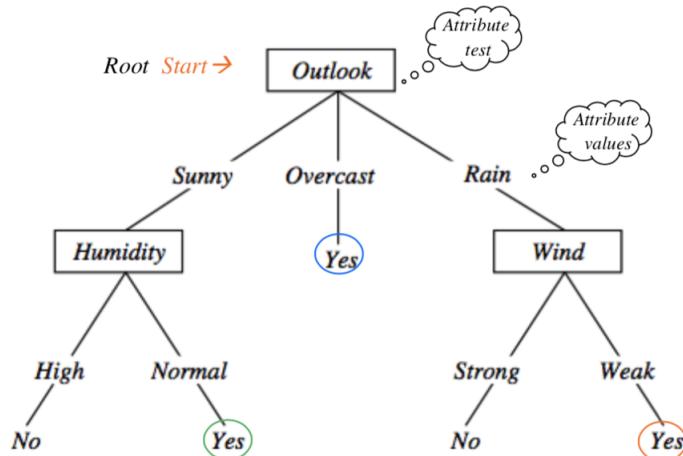
Un albero di decisione rappresenta una funzione che prende in input un vettore di attributi e restituisce una "decisione" (un singolo valore in output). I valori sia in input che in output possono essere valori continui, ma per adesso vedremo solo su valori discreti. Gli alberi di decisione aiutano a superare la restrizione della descrizione delle ipotesi solo tramite regole congiuntive (and). Ogni nodo dell'albero rappresenta un test su un attributo e ogni ramo corrisponde alla scelta del valore per quell'attributo, mentre le foglie corrispondono ad una classificazione.

### 6.1 Problema del PlayTennis

Vogliamo capire in quali giorni al nostro amico piace andare a giocare a Tennis. Il nostro Training Set è il seguente:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Il nostro albero di decisione ha invece questa forma:



L'albero rappresenta una disgiunzione di congiunzioni dei vincoli sui valori degli attributi, dove le foglie rappresentano la classificazione ottenuta. In particolare il nostro albero rappresenta questa formula:

$$\begin{aligned} & (\text{Outlook}=\text{Sunny} \wedge \text{Humidity} = \text{Normal}) \vee \\ & (\text{Outlook}=\text{Overcast}) \vee \\ & (\text{Outlook}=\text{Rain} \wedge \text{Wind}=\text{Weak}) \end{aligned}$$

Possiamo, in altro modo, definire l'albero come un insieme di regole if-then-else.

## 6.2 (Alg. ID3) Induzione Top-Down sugli alberi di decisione

L'algoritmo ID3 è un algoritmo base per l'apprendimento che sfrutta gli alberi di decisione. Dato un insieme di esempi, l'algoritmo per la costruzione dell'albero di decisione ha un approccio Top-Down eseguendo una ricerca Greedy senza backtracking. La domanda che ci dobbiamo porre quando stiamo creando l'albero è: "qual è il prossimo attributo da testare?" che tradotto in termini di alberi di decisione è scegliere il prossimo nodo che ci restituisce più Information Gain (definito successivamente). Viene quindi creato un nodo discendente per ogni possibile valore dell'attributo e gli esempi vengono partizionati in base a quel valore. Il processo viene ripetuto per ciascun nodo successore fino a quando tutti gli esempi sono stati classificati correttamente o non sono rimasti attributi.

```
ID3(X, T, Attrs)
    X: training examples
    T: target attribute
    Attrs: other attributes, initially all attributes

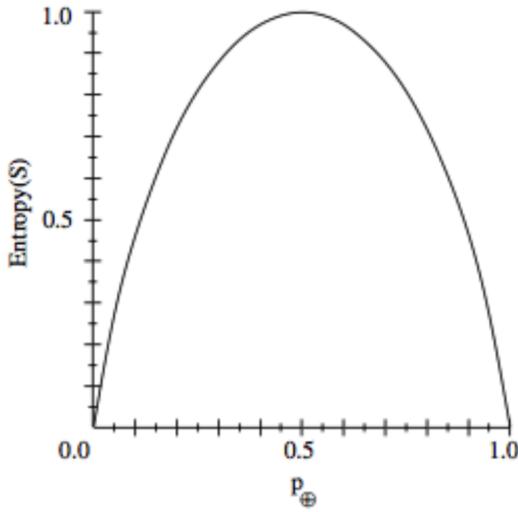
Create Root node
If all X's are +, return Root with class +
If all X's are -, return Root with class -
If Attrs is empty return Root with class most common value of T in X
else
    A ← best attribute; decision attribute for Root ← A
    For each possible value  $v_i$  of A:
        - add a new branch below Root, for test  $A = v_i$ 
        -  $X_i \leftarrow$  subset of X with  $A = v_i$ 
        - If  $X_i$  is empty then add a new leaf with class the most common value of T in  $X_i$ 
        else add the subtree generated by ID3( $X_i$ , T, Attrs - {A})
    return Root
```

## 6.3 Entropia: Come scegliere il miglior Attributo

La scelta principale in ID3 riguarda quale attributo testare ad ogni nodo. Per definire l'Information Gain dobbiamo usare la nozione di entropia, la quale misura l'impurità di una collezione di esempi.

- S è una collezione di esempi
- $p_+$  è una porzione di esempi positivi in S
- $p_-$  è una porzione di esempi negativi in S

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



Se gli esempi sono tutti positivi o tutti negativi (quindi appartengono tutti alla stessa classe) otteniamo una entropia che vale 0. Otteniamo una entropia massima se gli esempi positivi e negativi sono in ugual dimensione ( $\frac{1}{2}+, \frac{1}{2}-$ ).

Prendiamo come esempio  $S=14$  attributi:

$$Entropy([14+, 0-]) = -\frac{14}{14}log_2(\frac{14}{14}) - 0log_20 = 0$$

$$Entropy([7+, 7-]) = -\frac{7}{14}log_2(\frac{7}{14}) - \frac{7}{14}log_2(\frac{7}{14}) = 1$$

$$Entropy([9+, 5-]) = -\frac{9}{14}log_2(\frac{9}{14}) - \frac{5}{14}log_2(\frac{5}{14}) = 0.94$$

Sia l'entropia che la variabile  $p$  hanno valori tra lo 0 e 1 compresi. Una interpretazione dell'entropia è che essa specifica il numero minimo di bit necessari per rappresentare la classificazione di un membro random di  $S$ , per questo si utilizza il logaritmo in base 2.

## 6.4 Information Gain

L'Information Gain misura la riduzione aspettata dell'entropia causata dal partizionamento degli esempi rispetto all'attributo. Cioè:

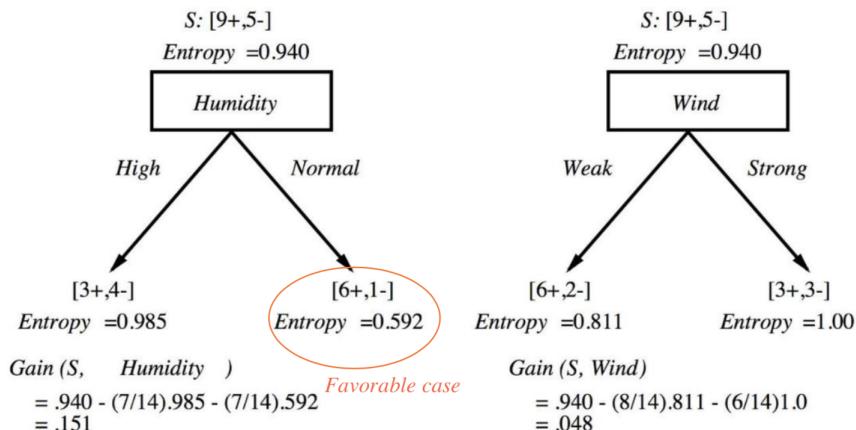
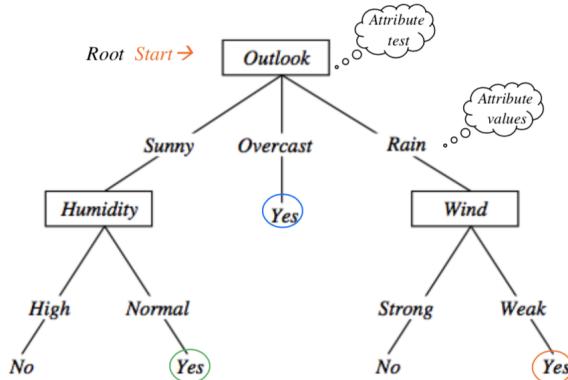
$$Gain(S, Attr) = Entropy(S) - \sum_{v \in Values(Attr)} \frac{|S_v|}{|S|} Entropy(S_v) \quad (23)$$

dove  $Values(Attr)$  è l'insieme dei possibili valori associabili all'attributo, mentre  $S_v$  è il sottoinsieme degli esempi  $S$  per i quali l'attributo ha valore  $v$ .

Notare come il primo membro dell'equazione è l'entropia su tutto l'insieme  $S$ , mentre il secondo è l'entropia attesa dopo che  $S$  è stato partizionato secondo l'attributo  $Attr$ . Maggiore è l'Information Gain, più efficace è l'attributo nella classificazione dei dati di training. Usiamo l'entropia per misurare l'omogeneità della classe del sottoinsieme degli esempi, quindi dobbiamo scegliere  $Attr$  in modo tale da massimizzare il Gain. Stiamo sottraendo  $\sum_{v \in Values(Attr)} \frac{|S_v|}{|S|} Entropy(S_v)$  all'entropia generale, ci aspettiamo di ottenere una maggiore omogeneità e quindi un Information Gain maggiore. Lo scopo è quello di separare gli esempi sulla base del target, trovando l'attributo che discrimina gli esempi che appartengono a classi diverse.

#### 6.4.1 Esempio

L'Information Gain viene calcolato per tutti gli attributi, e ID3 lo sfrutta (prende quello con Inf. Gain maggiore) per scegliere il prossimo attributo da testare. Nel nostro esempio  $\text{Gain}(S, \text{Outlook})=0.246$  ed è quello con information gain maggiore rispetto a agli altri 3. Quindi per il nodo radice è stato scelto l'attributo Outlook. I dati vengono partizionati rispetto ai valori di Outlook e associati ai nodi successivi. Notiamo che se Outlook=Overcast la risposta è sempre YES, quindi andiamo a valutare  $\text{Gain}(S_{\text{Sunny}}, \text{Humidity})$ ,  $\text{Gain}(S_{\text{Sunny}}, \text{Temperature})$  e  $\text{Gain}(S_{\text{Sunny}}, \text{Wind})$ . Trovando che il massimo gain si ottiene con Humidity. Se l'entropia non è zero l'albero continua a crescere fino ai nodi di classificazione.



Humidity fornisce più informazioni di Wind

#### 6.5 Problemi con Information Gain

L'information Gain favorisce gli attributi con molti valori possibili. Mettiamo caso che nel nostro problema PlayTennis ci fosse stato un attributo che indicava il giorno e il mese, ogni giorno a quel punto sarebbe stato un insieme puro con entropia 0 e quindi avrebbe avuto l'Information Gain più alto di tutti gli altri attributi. Facendo così sarebbe scelto come nodo radice, di conseguenza otterremmo un albero di profondità 1. Ma questo mi è inutile poiché il giorno non è un valore significativo per classificare le istanze future che ancora non abbiamo visto. Quindi come evitiamo la creazione di piccoli sottoinsiemi che mi riducono la generalizzazione?

## 6.6 Gain Ratio

$$GainRatio(S, Attr) = \frac{Gain(S, Attr)}{SplitInformation(S, Attr)} \quad (24)$$

dove

$$SplitInformation(S, Attr) = - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \quad (25)$$

dove  $c$  é il numero di variabili possibili per quell'attributo.  $S_i$  sono gli insiemi ottenuti partizionando sul valore  $i$  di Attr.

$SplitInformation$  misura l'entropia di  $S$  rispetto ai valori di A. Più i dati sono dispersi in modo uniforme, più è alto. GainRatio penalizza gli attributi che dividono gli esempi in tante piccole classi come per l'esempio dell'attributo della data. Sia  $|S| = numerodate$  divide gli esempi in  $n$  sottogruppi.

$$SplitInformation(S, Date) = -[(\frac{1}{n} \log_2 \frac{1}{n}) + \dots + (\frac{1}{n} \log_2 \frac{1}{n})] = -\log_2 \frac{1}{n} = \log_2 n \quad (26)$$

Il problema che può nascere é che lo  $SplitInformation$  può essere 0 o molto piccolo quando  $|S_i|$  é circa  $|S|$  per alcuni valori  $i$ . Ad esempio in un caso estremo in cui  $|S_1| = 0$  e  $|S_2| = 1$  otteniamo uno  $SplitInformation = -[0/n * log0/n + n/n * logn/n] = -0 - 0 = 0$ . Per evitare questo effetto viene utilizzata la seguente euristica:

1. Calcolare il Gain per ogni attributo
2. Applicare il GainRatio solo a quegli attributi con il Gain superiore alla media

## 6.7 Considerazioni sulla ricerca nello Spazio delle Ipotesi (in DT Learning)

Nell'algoritmo ID3 la ricerca nello spazio consiste nella ricerca dell'albero dal più semplice al più complesso. Mettendolo a confronto con l'algoritmo Candidate Elimination troviamo che:

- Lo spazio delle ipotesi é completo perché rappresenta tutte le funzioni a valori discreti relative agli attributi disponibili (mentre prima avevamo solo funzioni rappresentate da and)
- La ricerca mantiene solo una singola ipotesi corrente perché è un albero di decisione (mentre prima si manteneva il set di tutte le ipotesi consistenti)
- ID3 nella sua forma pura non utilizza backtracking e non c'è garanzia di trovare l'ottimo (perché è un algoritmo greedy e potrebbe fermarsi sui minimi locali)
- Usa tutti gli esempi a disposizione (mentre prima si guardava un esempio alla volta, in ID3 si usa tutto l'insieme degli esempi, siamo meno suscettibili all'errore sul singolo).
- Può terminare prima accettando classi rumorose (mentre prima non accettavamo classi rumorose)

## 6.8 Inductive Bias in DT Learning

Qual é il nostro Inductive Bias negli alberi di decisione? Cioè qual é il modo che ha ID3 per generalizzare dagli esempi di training?

L'algoritmo ID3 seleziona preferibilmente alberi corti rispetto a quelli con molti livelli e quelli che hanno un Information Gain alto vicino alla radice. Gli alberi di decisione non sono limitati nel rappresentare tutte le possibili funzioni, la restrizione non riguarda lo spazio delle ipotesi ma riguarda la strategia di ricerca.

Abbiamo due tipi di Bias:

1. Bias di ricerca: è legato alla strategia di ricerca. ID3 ricerca uno spazio completo di ipotesi; la strategia di ricerca è incompleta poiché essendo un algoritmo Greedy una volta che scelgo il primo attributo sto eliminando tutte le ipotesi successive che non corrispondono.
2. Bias di linguaggio: è legato all'insieme delle ipotesi esprimibili o considerabili. Candidate Elimination cerca uno spazio di ipotesi incompleto ma la strategia di ricerca è completa.

Perché preferiamo un Bias di ricerca? Se iniziamo con uno spazio di ricerca troppo stringente potremmo escludere la funzione target. Conviene avere uno spazio più ampio che poi andiamo a restringere con la strategia di ricerca <sup>4</sup>. In ML generalmente si usano approcci flessibili (capacità universale dei modelli), senza escludere a priori la funzione target sconosciuta ma ovviamente, flessibile → Overfitting!

## 6.9 Problemi con gli alberi di decisione

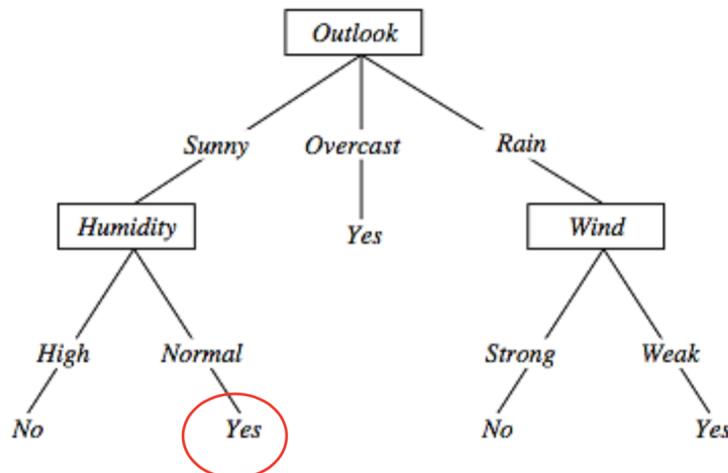
Il problema principale che può presentarsi quando si usano gli alberi di decisione è l'Overfitting (errore di taglio ridotto). Costruire alberi che si adattano troppo agli esempi di training portano all'Overfitting. Definiamo l'errore di una ipotesi  $h$  su:

- Dati di training:  $error_D(h)$  // errore empirico
- Tutti i dati contenuti in  $X$ :  $error_X(h)$  // errore atteso

L'ipotesi  $h$  "overfitta" i dati di allenamento se c'è un'altra ipotesi  $h'$  tale che:  
 $error_D(h) < error_D(h')$  e  $error_X(h') < error_X(h)$

detto a parole l'ipotesi  $h$  si comporta meglio sul singolo dato ma peggio sull'insieme di dati. Questo comporta che  $h'$  funziona male sul training set, ma molto meglio sui dati futuri non visti precedentemente.

Un esempio con un dato rumoroso è il seguente:  $\{Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, PlayTennis = No\}$  da come valore target NO, invece rispetto al nostro albero di decisione la risposta dovrebbe essere YES.



Questo nuovo esempio rumoroso provoca la divisione del secondo nodo foglia creando un albero più complesso. Adattandosi anche al rumore, il nuovo DT è perfetto per i dati di training ma non per i nuovi dati.

---

<sup>4</sup>Rasoio di Occam: "prefer the simplest hypothesis that fits the data"

### 6.9.1 Evitare l'Overfitting

Ci sono diversi approcci per evitare l'overfitting che possono essere racchiusi in due classi:

1. fermare la crescita dell'albero prima che abbia una classificazione perfetta
2. permettere all'albero di overfittare i dati e successivamente eseguire un post-potatura dei rami

Come possiamo valutare l'effetto di queste decisioni?

- Possiamo dividere il training set in due parti (training e validazione) e utilizzare il set di validazione per valutare l'utilità della post potatura
- Possiamo utilizzare tutti i dati di training e in seguito eseguire un test statistico per stimare l'effetto di espansione o potatura
- Possiamo usare un meccanismo di misurazione della complessità per la codifica degli esempi di allenamento e dell'albero. Cioè usare il principio della lunghezza minima della descrizione e fermare la crescita dell'albero quando la dimensione dell'encoding è minima.

### 6.9.2 Potatura con errore ridotto

Come possiamo usare il validation set per evitare l'overfitting? Nel reduced error pruning ogni nodo è un candidato per la potatura: la potatura consiste nella rimozione di un sottoalbero radicata in un nodo, quest'ultimo diventa una foglia e viene assegnata la classificazione più comune. I nodi vengono rimossi solo se l'albero risultante non ha prestazioni peggiori sul set di validazione. I nodi vengono eliminati in modo iterativo: ad ogni iterazione viene eliminato il nodo la cui rimozione aumenta la precisione del set di validazione. La potatura si interrompe quando nessuna potatura aumenta la precisione. Uno svantaggio è sacrificare dei dati del training set per formare il validation set.

### 6.9.3 Regola della post-potatura

Nella pratica un metodo che funziona spesso è la post-pruning rule. Le seguenti regole sono solo euristiche: non garantiscono l'ottimo a priori.

1. Si crea l'albero di decisione a partire dal training set fino a che i dati di training fittano al loro meglio. (stiamo facendo overfitting appositamente)
2. Si converte l'albero in un insieme equivalente di regole
  - Ogni path corrisponde a una regola
  - Ogni nodo lungo un path corrisponde a una pre-condizione
  - Ogni nodo foglia classifica

per esempio  $(Outlook = Sunny) \wedge (Humidity = High) \rightarrow (PlayTennis = No)$

3. Potare (generalizzare) ogni regola rimuovendo quelle pre-condizioni la cui rimozione migliora l'accuracy sia sul validation set, sia sul training set.
4. Ordinare le regole in ordine di precisione stimato e considerarle in sequenza quando si classificano nuove istanze

Perché trasformare ogni path in regola? Ogni percorso distinto produce una regola diversa, la rimozione di una condizione può essere basata su un criterio locale. La potatura delle pre-condizioni è specifica delle regole mentre la potatura dei nodi è globale e influisce su tutte le regole!

## 6.10 Problema: Valori continui degli attributi

Fino ad ora abbiamo usato valori discreti per gli attributi, come possiamo affrontare quelli a valori continui?

Dato un attributo a valore continuo  $A$ , si crea dinamicamente un nuovo attributo  $A_c$  tale che:  $A_c = True$  se  $A < c$ , False altrimenti.

Ma come determinare il valore di soglia  $c$ ? Un esempio sul tennis potrebbe essere:

<i>Temperature</i>	40	48		60	72	80		90
<i>PlayTennis</i>	No	No	<i>54</i>	Yes	Yes	Yes	<i>85</i>	No

Cioè possiamo determinare il valore utilizzando la media tra due esempi consecutivi dove però abbiamo un cambio di classificazione: nell'esempio  $(48+60)/2=54$  e  $(80+90)/2=85$ .

## 6.11 Problema: Dati di training incompleti

Come possiamo invece risolvere il problema dei dati mancanti? La strategia è utilizzare altri esempi per "indovinare" l'attributo in due modi:

- Si assegna il valore più comune tra tutti gli esempi di training sul nodo o quelli della stessa classe.
- Si assegna una probabilità a ciascun valore possibile da assegnare, in base alle frequenze, si associa il valore all'attributo mancante secondo la distribuzione di probabilità.

I valori mancanti in nuove istanze da classificare vengono trattati di conseguenza e viene scelta la classificazione più probabile.

## 6.12 Problema: attributi con costi differenti

In alcuni problemi le istanze degli attributi possono avere un costo associato, cioè possiamo dare più importanza ad alcuni attributi rispetto ad altri. Preferiamo gli alberi che usano costi bassi per gli attributi. L'algoritmo ID3 può essere modificato per lavorare anche con i costi:

- Tan e Schlimmer

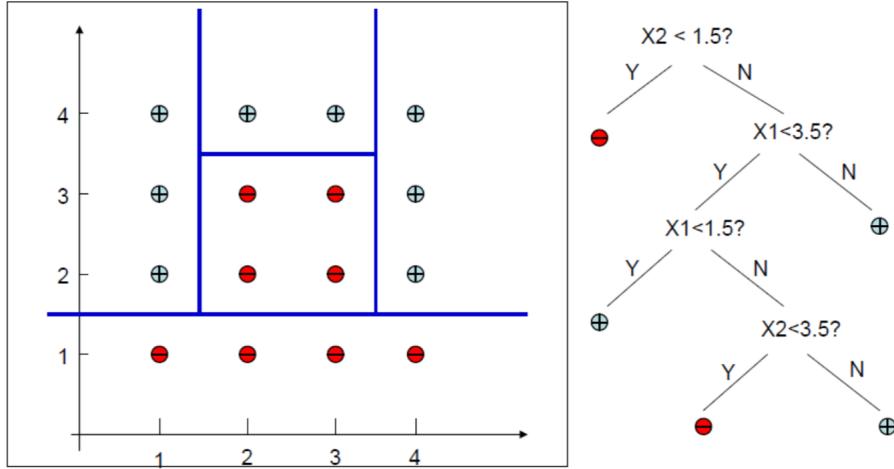
$$\frac{Gain^2(S, Attr)}{Cost(S, Attr)} \quad (27)$$

- Nunez

$$\frac{2^{Gain(S, Attr)} - 1}{(Cost(Attr) + 1)^w} \text{ con } w=0/1 \quad (28)$$

### 6.13 Visione Geometrica (Decision Boundaries dei DT)

I decision boundaries, che possono essere prodotti da un albero di decisione, dividono lo spazio di input in rettangoli paralleli agli assi ed etichettano ogni rettangolo con una delle classi K (foglia dell'albero).



### 6.14 Conclusioni sugli alberi di decisione

Gli alberi di decisione sono un approccio molto utilizzato per la classificazione con un numero discreto di classi. Lo spazio delle ipotesi comprende l'area degli approcci proposizionali quindi basati su regole. Si comporta bene con dati rumorosi e gestisce anche i valori degli attributi mancanti. È facile da capire (regole if-then-else, a meno che non lo siano troppe). ID3 ricerca in uno spazio completo di ipotesi, con una strategia greedy INcompleta (perché manca il backtracking, una volta che fissa il primo nodo, non torna indietro). Presenta un approccio flessibile: l'overfitting è un problema importante che affrontata con il post-potatura e la generalizzazione delle regole indotte dall'albero.

## 7 Validation

Un problema fondamentale del ML è valutare la capacità di generalizzazione del nostro modello, una domanda che dobbiamo porci è: quando un modello è un buon modello? L'apprendimento, come abbiamo visto, consiste nel trovare una buona funzione in uno spazio di funzioni costruito a partire dai dati conosciuti. "Buona" in relazione al fatto di effettuare un basso errore di generalizzazione, per sapere quanto accuratamente il modello si comporta su nuovi dati non visti precedentemente.

In particolare, qualsiasi  $h$  (ipotesi) che approssima bene  $f$  (la nostra funzione da cercare) sui dati di training, approssimerà bene anche su istanze non conosciute? La generalizzazione è quindi un punto cruciale del ML, dobbiamo trovare un modo in cui scegliamo il miglior modello tra svariati modelli calcolati. Abbiamo visto che si procede con una fase di apprendimento (Learning phase) dove si costruisce (training, fitting) il modello sfruttando i dati conosciuti. Segue poi una fase predittiva (Predictive phase) dove si applica il modello a nuovi dati sconosciuti e si valuta l'ipotesi predittiva (ad esempio si valuta la capacità di generalizzazione), quest'ultima è una fase di test.

Per valutare generalmente usiamo due misurazioni:

- Per la classificazione: Mean Square Error sulla funzione di Loss e l'accuratezza o Mean Error Rate per il risultato.
- Per la regressione: Mean Square Error o Mean Absolute Error...

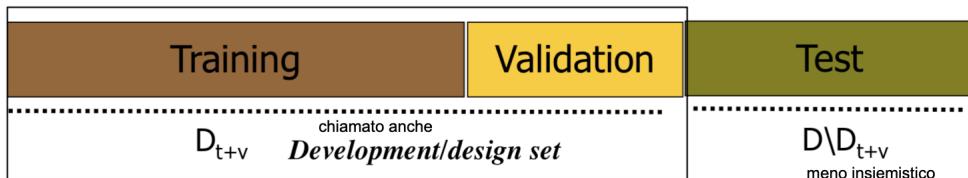
## 7.1 Obiettivi Validazione

La validazione ha due obiettivi principali:

- Model Selection: stima la performance (errore di generalizzazione) di differenti modelli di apprendimento in modo tale da scegliere il migliore. Il compito di selezione del modello ovviamente restituisce un modello.
- Model Assessment: dopo aver scelto un modello finale, stimiamo/valutiamo il suo errore/rischio di previsione (errore di generalizzazione) su nuovi dati di test mai visti prima (magari dati dal cliente). Il compito di valutazione di un modello restituisce una stima.

## 7.2 Holdout Cross-Validation

Per ottenere una valutazione accurata di un modello dobbiamo valutarlo su esempi nuovi mai visti. Ma noi abbiamo solo un insieme dati limitati e non dobbiamo sprecarli. Dobbiamo quindi suddividere il nostro data set  $D$  in Training Set, Validation Set e Test Set.



Il Training Set verrà usato dall'algoritmo per la creazione dell'ipotesi  $h$ . Il Validation Set viene usato per scegliere il miglior modello tra differenti modelli costruiti. Infine usiamo il Test Set solo per la valutazione del modello.

Cosa succede se il test set viene utilizzato in un ciclo ripetuto di design? Stiamo eseguendo una selezione del modello e non una valutazione dell'errore di generalizzazione affidabile. Purtroppo non saremo in grado di valutare il modello su esempi non visti, poiché ormai "ce li siamo bruciati". In questo caso, dato che abbiamo usato il test set abbiamo una valutazione sovraottimistica. La regola d'oro è tenere una separazione tra i compiti e usare insiemi separati per Training, Validation e Test.

## 7.3 Meta-Algoritmo per l'utilizzo del Data Set

- separare TR, VL e TS
- cercare la migliore ipotesi  $h_{w,\lambda}$  cambiando gli iper-parametri  $\lambda$  del modello
- per ogni valore differente dell'iper-parametro  $\lambda$ : cercare la migliore ipotesi che minimizza l'errore / perdita empirica (fittando i dati sul TR set) trovando i migliori parametri  $w$ .
- selezionare la migliore ipotesi  $h_{w,\lambda}$  dove migliore significa con il minor errore sul Validation Set!
- opzionale: è possibile far fittare  $h$  su TR+VL con  $\lambda$  ormai fissato
- valutare la  $h$  finale sul Test Set

La ricerca dei migliori iper-parametri può consistere in una ricerca in una griglia di valori candidati, per ogni modello calcolare il risultato su Validation Set e prendere quello che con il minimo errore a massima accuratezza.

Hyper-param.	Lambda 0.1	Lambda 0.01	Lambda 0.001
Degree 1	Res1	Res4	Res7
Degree 2	Res2	Res5	Res8
Degree 4	Res3	Res6	Res9

Nell'immagine Res1 è calcolato sul Validation Set dal modello con un polinomio di grado 1 e un  $\lambda = 0.1$  (modello allenato sul training set). Il migliore è Res3, perché ha un polinomio di grado 4 e  $\lambda = 0.1$ . Questi calcoli sono facili da parallelizzare.

### 7.3.1 Esempio

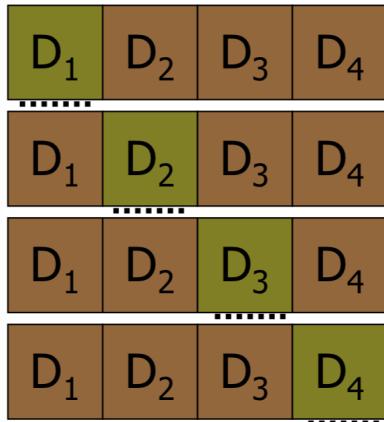
$\lambda$	TR	VL	TS
0.5	75	70	70
0.1	80	75	70
0.01	90	70	72

- In che ordine si usano le porzioni di dati per calcolare i valori in tabella?  
Prima il training, poi validation e infine il test
- Quale modello (ossia lambda) si sceglie?  
Si sceglie il migliore sul Validation Set, NON si usa il Test Set per il model selection!

### 7.3.2 Esempio perché separare VL e TS

Ci troviamo in una situazione con 20-30 esempi, 1000 variabili di input, come output abbiamo target 0/1 calcolato randomicamente. Trovo un modello con una sola variabile che indovina "per caso" al 99% su training e validation set. Otteniamo un risultato perfetto (cioè un modello con accuratezza al 99%)? 99% non è una buona stima dell'errore di test (quella corretta è 50% visto che gli 0/1 in output sono randomici). L'errore stimato su training o validation per il model selection NON è utile! Usare tutto il data set per feature/model selection lede la correttezza della stima. Se effettuo la selezione del modello su tutto il data set e poi eseguo training e validation su sottoinsiemi, il test fornirà risultati biased (FS-bias). Un test set esterno fornisce invece la stima corretta del 50%.

## 7.4 K-fold cross validation



Hold out cross validation, che abbiamo visto prima, può effettuare un uso insufficiente di dati. Il seguente metodo ogni volta riparte da zero e non tiene conto di quello che è successo prima.  
K-fold cross validation:

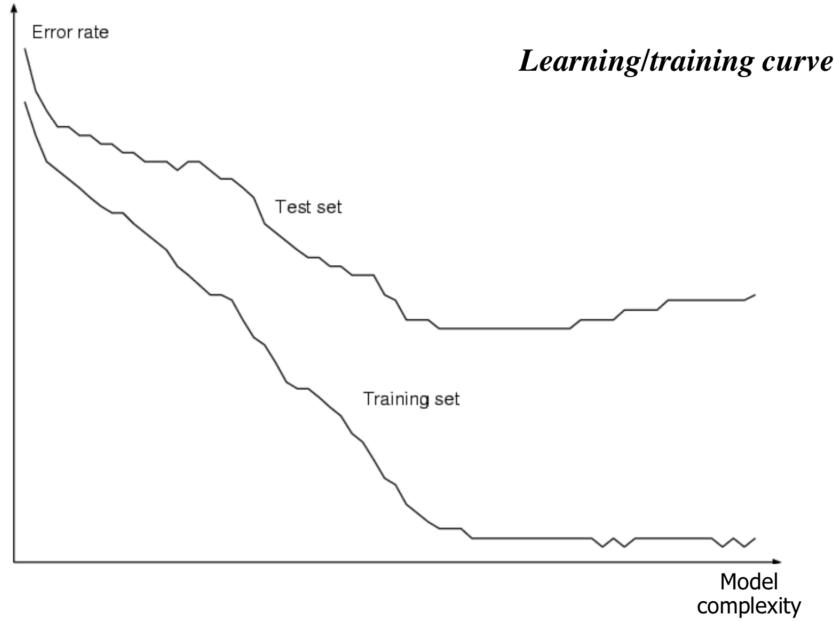
Scegliere un  $\lambda$ . Dividere il data set in  $D$  porzioni mutualmente esclusive di dati. Allenare l'algoritmo di apprendimento sull'insieme  $D - D_i$  e testarlo su  $D_i$ . Calcolo la media sui risultati trovati su  $D_i$ . Cambio  $\lambda$  e rieseguo il tutto. Alla fine scelgo il modello con il minimo errore di validation, sulla base del valore calcolato con la media sui vari  $D_i$ .

Utilizza tutti i dati per training, validazione o test. NOTA: questa tecnica può essere utilizzata sia per il validation set, sia per il test set. In quante parti dobbiamo dividere? 3,5,10... ma spesso è computazionalmente costoso.

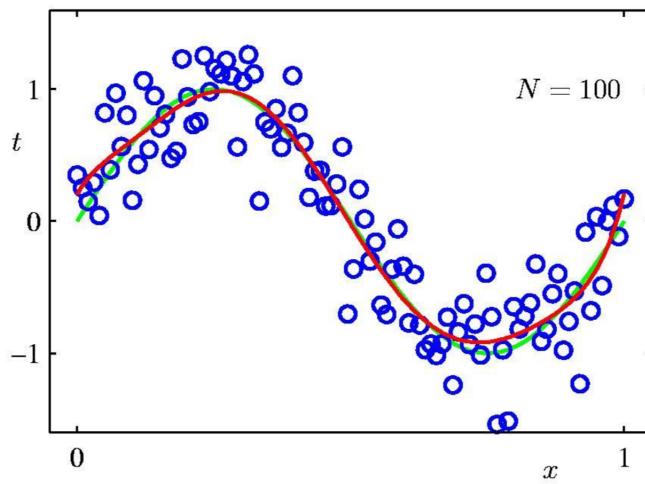
## 7.5 Esempio di Model Selection e Assessment

Dividere i dati in Training Set e Test Set (sfruttando Hold out o K-fold), usare K-fold internamente sul Training Set per trovare i migliori iper-parametri  $\lambda$  del modello. Eseguire una ricerca su griglia con tutti i possibili valori dell'iper-parametro e scegliere il miglior  $\lambda$ . A questo punto allenare su tutto il TR set il modello finale e valutarlo su un test set esterno.

## 7.6 Comportamento tipico di un algoritmo di apprendimento



All'inizio sia l'errore di training che quello di test sono alti, perché? Succede un po' quello che succedeva con il grado del polinomio troppo basso, che non fittava bene i dati → Underfitting.  
 Alla fine invece l'errore di training è basso mentre l'errore sul test è alto, perché? È il caso in cui si sta overfittando sui dati di training e quindi il grado del polinomio è molto alto, ma non si sta comportando bene riguardo la generalizzazione sui dati di test sconosciuti.



Se prendiamo un polinomio con grado alto, ma abbiamo molti dati in più, predico meglio la funzione target. Ma quindi il problema era l'alto grado del polinomio o i pochi dati a disposizione? Per dare una risposta a questa domanda esiste della teoria di supporto.

## 7.7 Statistical Learning Theory

Mettendo insieme:

- La capacità di generalizzazione (misurata come errore di test) di un modello rispetto all'errore di training, delineando le zone di overfitting o underfitting
- Il ruolo della complessità del modello
- Il ruolo del numero di dati

troviamo la Statistical Learning Theory (SLT):

- si approssima una funzione sconosciuta  $f(x)$
- si cerca di minimizzare la funzione di rischio  $R = \int Loss(t, h(x)) dP(x, t)$  (Integrale della Loss su tutti i dati).
- vengono dati un valore target  $t$ , una probabilità di distribuzione  $P(x, t)$  e una funzione Loss del tipo  $L(h(x), t) = (t - h(x))^2$
- il compito finale è quello di cercare una ipotesi  $h$  nello spazio delle ipotesi  $H$ , dove il valore di rischio generale  $R$  è minimo, ma noi abbiamo solo un data set finito  $TR(x_i, t_i)$  con  $i=1\dots l$
- per cercare  $h$  dobbiamo quindi minimizzare l'errore empirico (training error) trovando il miglior valore per il modello con parametri liberi  $w$ .

$$R_{emp} = \frac{1}{l} \sum_{i=1}^l (t_i - h(x_i))^2$$

Questo si chiama principio induttivo della minimizzazione del rischio empirico (Empirical Risk Minimization). La domanda che dobbiamo porci è: possiamo usare  $R_{emp}$  per approssimare  $R$ ?

## 7.8 Teoria di Vapnik-Chervonenkis + SLT

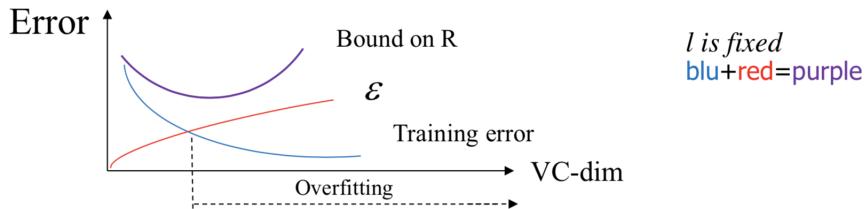
- VC-dimension è una misura per la complessità dello spazio delle ipotesi  $H$  (flessibilità per fittare i dati, come ad esempio il numero di parametri per modelli lineari/grado del polinomio)
- VC-bounds afferma che con una probabilità di  $1 - \delta$  che

$$R \leq R_{emp} + \epsilon\left(\frac{1}{l}, VC, \frac{1}{\delta}\right)$$

cioè che il rischio generale  $R$  ha un limite superiore definito dal rischio empirico (che decresce con la complessità dei modelli) più il VC-confidence. Il VC-confidence ( $\epsilon$ ) aumenta all'aumentare della complessità del modello (indicata da  $VC$ ) e diminuisce quando il numero dei dati aumenta. ( $1/\delta$  indica la probabilità che valga questo bound)

Possiamo usare il rischio empirico per approssimare  $R$ ? Sì, il Machine Learning è ben fondato, l'intuizione sta nel fatto che un numero alto di dati portano ad un  $R$  basso, mentre un alto  $VC$ -dim abbassa  $R_{emp}$  ma potrebbe alzare  $R$  (overfitting).

## 7.9 Structural Risk Minimization



Sfruttiamo il concetto di controllo della complessità del modello, eseguiamo un trade-off tra la complessità del modello e l'accuratezza sul TR set.

## 7.10 Conclusioni

La STL permette di inquadrare formalmente il problema della generalizzazione e overfitting, fornendo limitazioni superiori analitiche e quantitative al rischio  $R$  di predizione su tutti i dati, indipendentemente dal tipo di learning algorithm o dettagli del modello. Il ML è ben fondato, il rischio del learning (e dell'errore di generalizzazione) può essere analiticamente limitato! Si può trovare un buona approssimazione della  $f$  target da esempi, a patto di avere un buon numero di dati e una adeguata complessità del modello (misurabile formalmente con la VC-dim). Questo ci porta a nuovi modelli (SVM) (e altri metodi che direttamente considerino il controllo della complessità nella costruzione del modello). La STL fonda uno dei principi induttivi sul controllo della complessità.

Alcuni esempi di controllo della complessità sui modelli lineari sono il numero di parametri liberi  $w$  o la dimensione in input. Mentre per i Decision Trees il numero di nodi. Vedremo un approccio diretto alla complessità attraverso il modello SVM.

# 8 Support Vector Machine

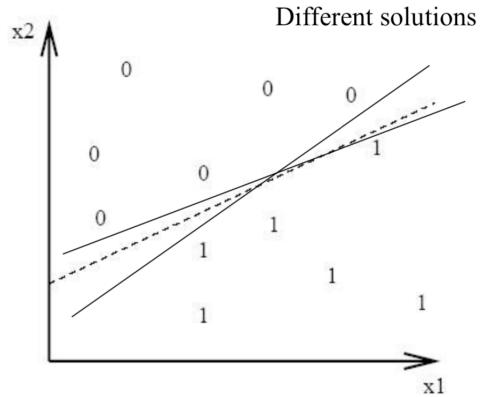
Il Support Vector Machine (o SVM) è l'approccio più famoso per il supervised learning. È un classificatore derivato dalla Statistical Learning Theory, dopo anni di sviluppi teorici, SVM è diventato famoso quando, usando immagini come input, dava una accuratezza comparabile a reti neurali.

### 8.0.1 Obiettivi utilizzo SVM

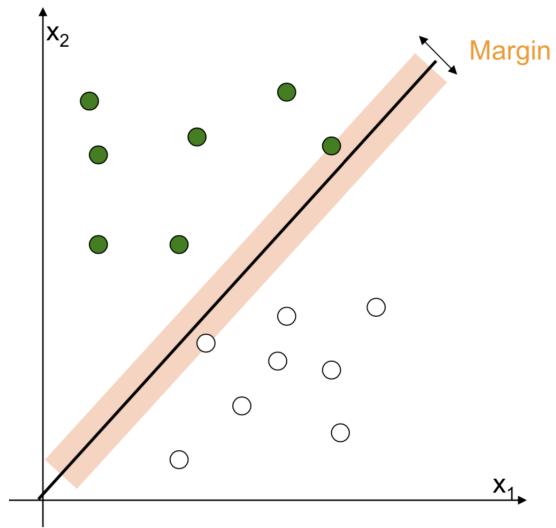
1. Controllare la complessità del modello attraverso un approccio di ottimizzazione, approssimando la minimizzazione del rischio strutturale (Max Margin Classifier).
2. Usare efficientemente l'espansione lineare via kernel in modo tale da ottenere un approccio flessibile per il Supervised Learning non lineare (Kernel).
3. Evitare di utilizzare SVM nel modo sbagliato.

## 8.1 Maximum Margin Classifier (controllo complessità)

Viene utilizzato nella classificazione binaria di problemi, mettiamoci nel caso in cui abbiamo un sistema linearmente separabile, non tutti gli iper-parametri che risolvono il problema sono uguali

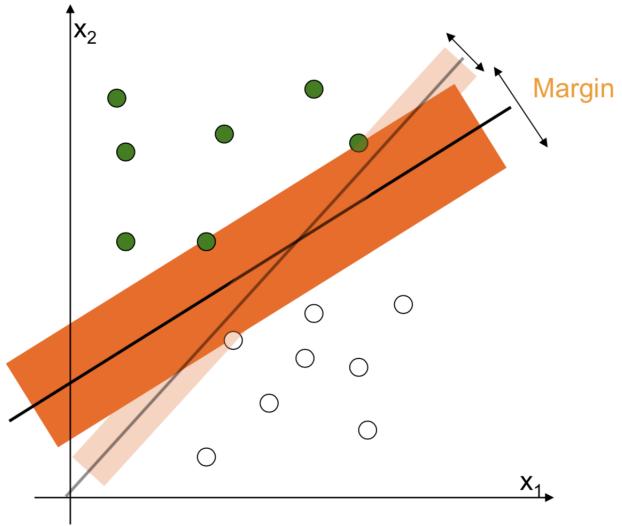


Grazie a SVM abbiamo un criterio per scegliere le soluzioni: minimizzare il rischio strutturale.



Il margine è (il doppio della) la distanza tra l'iperpiano e il dato più vicino.

Notiamo che però non tutti gli iperpiani che risolvono il problema sono uguali, variando l'iperpiano anche il margine cambia.



Ed è per questo motivo che cerchiamo il classificatore con il margine massimo!

## 8.2 Rappresentazione canonica dell'iperpiano e Support Vector

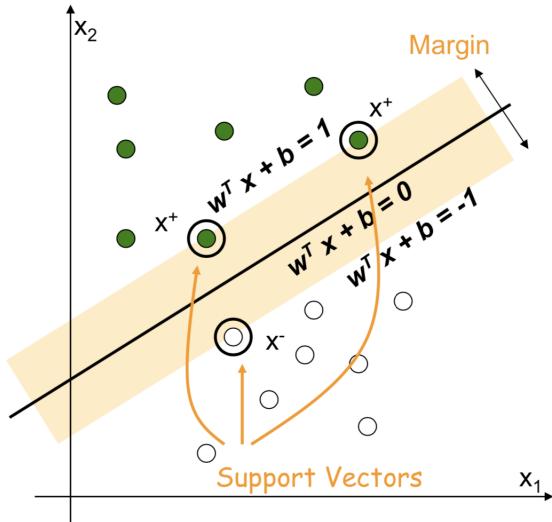


Figure 7: nota:  $w_0 = b$

$$\text{Support Vectors: } x_i : |w^T x_i + b| = 1$$

Tutti i punti sono classificati correttamente se  $(w^T x_i + b)y_i \geq 1 \quad \forall i$

### 8.3 Verso l'ottimizzazione del margine

Consideriamo il problema di apprendere un modello lineare per classificazione binaria  $h(x) = \text{sign}(wx + b)$ . Il problema consiste nel trovare il vettore  $w$  e il valore  $b$  in modo tale che tutti i punti sono classificati correttamente e il margine è massimizzato.

La rappresentazione canonica dell'iperpiano è definita come:

Il punto  $(x_i, y_i)$  è classificato correttamente per tutti gli  $i$  (tutti gli esempi del training set) se:

- $w^T x_i + b \geq 0$  se  $y_i = 1$
- $w^T x_i + b < 0$  se  $y_i = -1$

è possibile scalare  $w$  e  $b$  in modo tale che i punti più vicini all'iperpiano soddisfino  $|w^T x_i + b| = 1$  e quindi

- $w^T x_i + b \geq 1$  se  $y_i = 1$
- $w^T x_i + b \leq -1$  se  $y_i = -1$

Due fatti a noi utili sono:

1. Margine =  $2/|w|$  e sappiamo anche che  $|w|^2 = (w^T w)$ .  
Quindi per massimizzare il margine  $\rightarrow$  minimizzare  $|w| \rightarrow$  minimizzare  $|w|^2/2$
2. Il VC-dim del SVM è inversamente proporzionale al margine, quindi controlliamo la complessità del modello utilizzando il margine

Come abbiamo accennato precedentemente, l'iperpiano ottimo è l'iperpiano che massimizza il margine e ovviamente risolve il problema di training.

### 8.4 Problema di ottimizzazione quadratica

Il nostro problema è quindi diventato

minimizzare  $|w|^2/2$  in modo tale che  $(w^T x_i + b)y_i \geq 1$  per tutti gli  $i$ .

Questa è la nostra forma primale del problema, ne esiste anche una duale. Nota: cerchiamo la minimizzazione diretta della complessità del modello (funzione di ottimizzazione). Essendo il problema linearmente separabile, dobbiamo mantenere la soluzione tramite i vincoli. In relazione alla SLT minimizzando la VC-dim, diminuiamo anche il bound di errore.

### 8.5 Nuovo classificatore $h(\mathbf{x})$ (Problema Duale)

La soluzione ottima può essere trovata anche massimizzando

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \text{ con } \alpha_i \geq 0$$

Il nostro compito è cercare un vettore  $\alpha$  ottimale, dopo averlo trovato (calcolato dalla forma duale) possiamo calcolare i vettori  $w = \sum \alpha_i y_i x_i$  e  $b = y_k - w^T x_k$  per qualsiasi  $\alpha_k > 0$ . Possiamo adesso definirci un nuovo classificatore

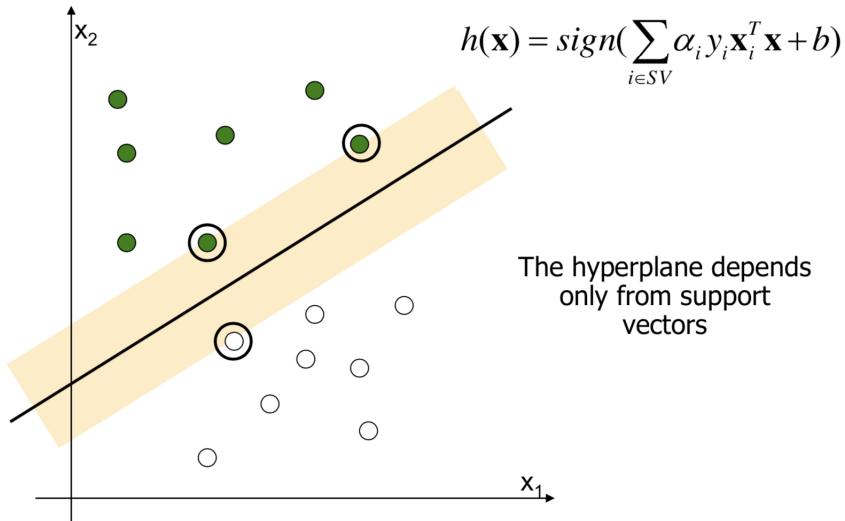
$$h(x) = \text{sign}(w^T x + b) = \text{sign}\left(\sum_{i=1}^l \alpha_i y_i x_i^T x + b\right) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i x_i^T x + b\right) \quad (29)$$

$x_i^T x$  è il prodotto scalare tra le  $x$  dei vettori di supporto (presi ovviamente dal training set) e la  $x$  da classificare.

Una proprietà importante è che

i pesi  $\alpha_i$  associati ad ogni dato sono diversi da zero ad eccezione dei Support Vector. Cioè se  $\alpha_i \neq 0$  allora  $x_i$  è un Support Vector.

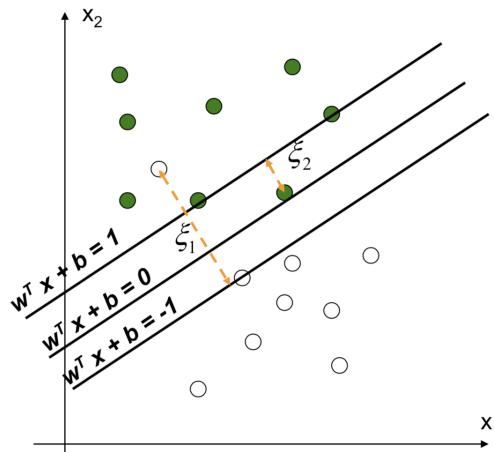
La soluzione è sparsa e formulata solo nei termini dei Support Vectors, in poche parole l'iperpiano dipende solo dai Support Vectors! Inoltre è una forma speciale della soluzione in cui non dobbiamo calcolare esplicitamente ( $w, b$ ) per classificare i punti.



## 8.6 Margine Soft

Purtroppo però, nella realtà, non abbiamo quasi mai dei problemi che sono linearmente separabili e con zero errore di classificazione. Anche se il problema fosse linearmente separabile, rischieremmo di avere un margine piccolissimo. Ammettiamo quindi errore introducendo le slack-variables (variabili di rilassamento).

Le variabili slack  $\xi_i$  possono essere aggiunte per permettere classificazioni errate o punti "molto rumorosi"



Modifichiamo il nostro problema in forma primale:

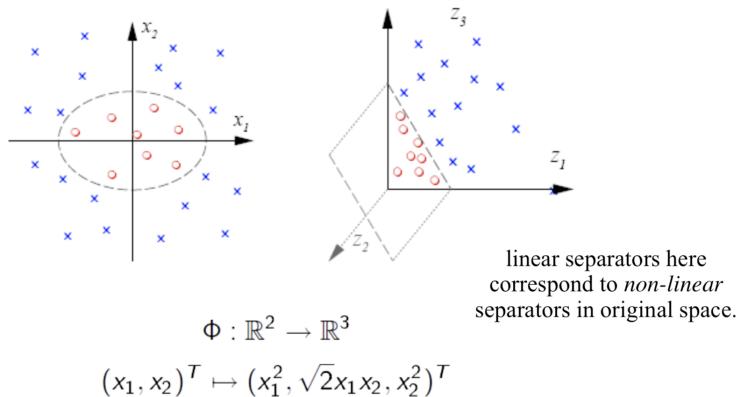
$$\text{minimizzare } |w|^2/2 + C \sum_i \xi_i \text{ tale che } (wx_i + b)y_i \geq 1 - \xi_i \text{ e } \xi_i \geq 0 \text{ per ogni } i$$

Possiamo vedere  $\xi$  come la distanza tra il vettore di supporto e il punto che stiamo analizzando, mentre  $C > 0$  gestisce il numero di errori permessi ( $C$  è un valore definito dall'utente). Con un  $C$  basso stiamo accettando tanti errori sul TR Set (perché gli  $\xi$  possono crescere molto) e quindi potremmo sfociare in un possibile underfitting. Con un  $C$  alto non permettiamo errori sul TR Set e quindi potremmo cascicare in overfitting.

## 8.7 Mapping per Spazi Dimensionali Ampi

Il compito di SVM è quello di creare un iperpiano che separa linearmente i dati, ma spesso non è possibile dividere linearmente un problema in una certo spazio dimensionale... ed è in questi casi che viene utilizzato il "kernel trick". Cioè spesso i dati non sono linearmente separabili nello spazio dimensionale in input, ma potrebbero essere divisibili in uno spazio dimensionale più grande.

Abbiamo visto cosa succede in casi non lineari: dobbiamo utilizzare efficientemente l'espansione della base via kernel in modo tale da ottenere un approccio flessibile anche per il Supervised Learning non lineare.



Ma questo lo sapevamo già, quando usavamo la funzione  $\phi(x)$  al posto di  $x$ .

$$h_w(x) = \text{sign}(\sum_k w_k \phi_k(x))$$

Sappiamo anche che l'utilizzo di spazi ad alta dimensione (funzioni di espansione della base) può essere non conveniente nel calcolo dal punto di vista computazionale e può facilmente portare a un overfitting, nel caso in cui non controlliamo la dimensione dello spazio e la complessità del classificatore (la complessità in questo caso è correlata alla dimensionalità dell'input).

Useremo l'approccio del Kernel per gestire (implicitamente) lo spazio nel contesto della modellizzazione regolarizzata (la complessità dipende dal margine). Pertanto, grazie alla regolarizzazione automatizzata di SVM, la complessità del classificatore può essere mantenuta ridotta indipendentemente dalla dimensionalità nel nuovo spazio.

In SVM non è necessario calcolare  $w$  e nemmeno calcolare direttamente la  $\phi$ . Poiché sfruttiamo il Kernel. Il Kernel è il risultato del prodotto scalare. Non ci importa di come è fatta la phi: non ho bisogno di espanderla per poi fare il prodotto scalare, ma mi basta il risultato di  $K(x_j, x_k)$ .

$$h(x) = \text{sign}(\sum_{i \in SV} \alpha_i y_i \phi(x_i)^T \phi(x)) = \text{sign}(\sum_{i \in SV} \alpha_i y_i K(x_i, x))$$

### 8.7.1 Esempi di Kernel

- Lineare:  $K(x_i, x_j) = x_i^T x_j$   
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  è x stesso
- Polinomiale:  $K(x_i, x_j) = (1 + x_i^T x_j)^p$  con p iper-parametro che indica il grado del polinomio  
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  ha dimensione esponenziale rispetto a p
- RBF (radial-basis-function) Gaussiana:  $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2}$  dove  $\sigma$  è un iper-parametro  
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  è di dimensione infinita  
(molto potente sul TR ma può portare ad overfitting)

## 8.8 Riassunto procedimento SVM con Kernel Fun.

- Scegliamo un parametro C (trade-off)
- Scegliamo una funzione di Kernel K (e i suoi parametri)
- Troviamo il miglior  $\alpha$
- Usiamo il modello finale

$$h(x) = \text{sign}(\sum_{i \in SV} \alpha_i y_i K(x_i, x))$$

## 8.9 Utilizzare bene SVM

Evitare errori di interpretazione tipici nell'uso di SVM. È possibile che si verifichi un overfitting senza un'attenta selezione dei parametri: C, funzione Kernel, parametri del Kernel, ecc... Il trattamento implicito dello spazio dimensionale ampio deve avvenire nello spazio delle caratteristiche e non in quello di input. Anche la tecnica di validation vista fino ad ora per la selezione del modello e la valutazione del modello devono essere utilizzate rigorosamente. Quindi:

- Trasformare i dati in un formato leggibile da un software SVM (ad esempio {red, green, blue} → (0,0,1), (0,1,0), (1,0,0))
- Eseguire una semplice scalatura dei dati (ad esempio in un intervallo da [-1,1] o [0,1])
- Considerare il kernel  $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2}$
- Usare la cross-validation per trovare i migliori parametri C e  $\sigma$
- Riutilizzare il TR set però con i migliori C e  $\sigma$  trovati
- Eseguire il testing su un Test Set esterno

## 9 K-Nearest Neighbors

K-NN fa parte del supervised learning, vediamo la forma piú semplice di questo metodo basato su istanze.

### 9.1 1-Nearest Neighbor

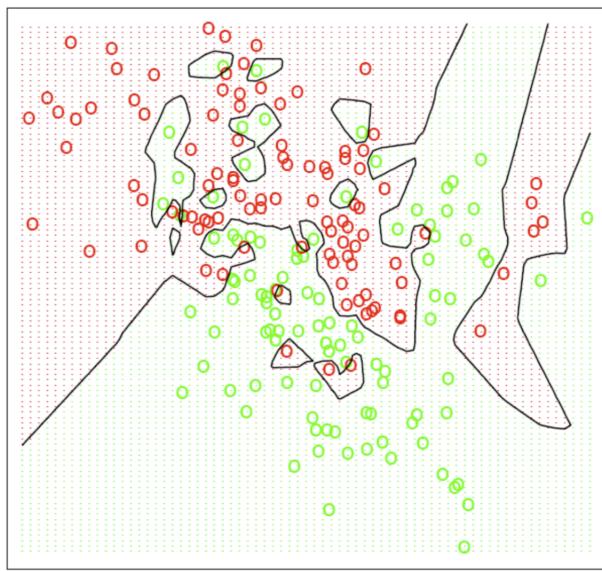
Questo algoritmo non impara, ma sfrutta tutti i valori del training set.

- Salvare i dati di training nella forma  $\langle x_j, y_j \rangle$  con  $j=1\dots l$
- Dato un input  $x$  di dimensione  $n$ :
  - Trovare l'esempio di training più vicino  $x_i$  tale che  $d(x, x_i)$  è minimo  
dove

$$d(x, x_j) = \sqrt{\sum_{t=q}^n (x_t - x_{jt})^2} = \|x - x_j\|$$

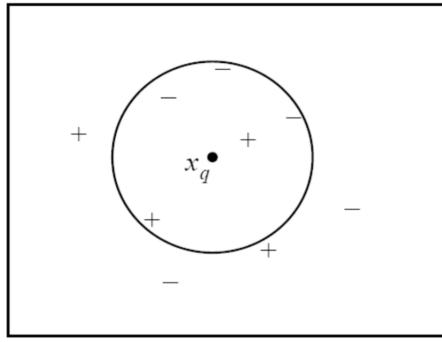
$x_t$  è la componente t-esima di  $x$ ,  $x_{jt}$  è la componente t-esima di  $x_j$

- Diamo come output  $y_i$  (in pratica stiamo vedendo l'esempio che piú assomiglia ai nostri dati e rispondiamo come ha risposto lui



È molto flessibile, non c'è errore di classificazione sui dati di TR, il decision boundary non è piú lineare e potrebbe portare all'overfitting?

## 9.2 K-NN



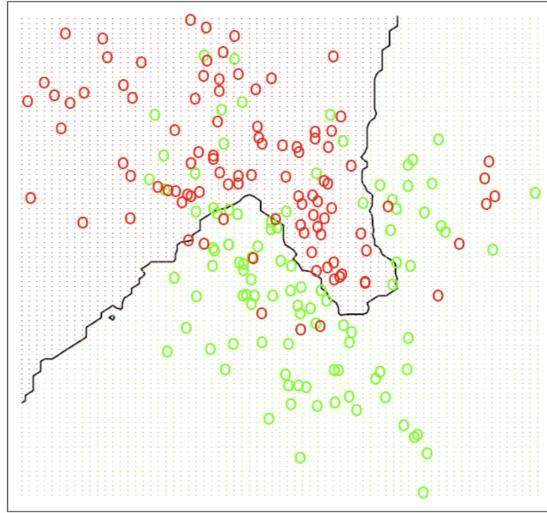
Vediamo già come si comporta in questo caso 1-nn risponderebbe + per  $x_q$ , valutando invece con 5-nn viene restituito - per  $x_q$ . Come succedeva con il polinomio di alto grado che andava in overfitting, anche qui dobbiamo renderlo più "smooth" valutando su un insieme di vicini.

Per questo possiamo "dare un occhiata" a tot (k) vicini e restituire una media:

$$avg_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

dove  $N_k(x)$  è il vicinato di  $x$  che contiene esattamente  $k$  vicini. Se c'è una chiara dominanza di una delle classi nel vicinato di  $x$ , allora è probabile che anche  $x$  appartenga a quella classe. Quindi la regola di classificazione è la maggioranza che vota tra i membri di  $N_k(x)$ .

$$h(x) = 1 \text{ se } avg_k(x) > 0.5 \text{ 0 altrimenti}$$



Dobbiamo trovare il giusto trade-off tra underfitting e overfitting trovando il giusto valore  $k$ . (Nota: è possibile usare k-nn anche se ci sono più classi).

### 9.3 Considerazioni su K-nn

Non c'è una ipotesi globale per tutte le istanze: non c'è quindi nessun modello da allenare. Dobbiamo semplicemente memorizzare gli esempi di input, inoltre il modello non è parametrico. Si tratta di una stima lineare globale della funzione target (sullo spazio dell'istanza). È un metodo basato sulla memoria, sull'istanza e sulla distanza, Bias induttivo! scegli il risultato in base alla distanza.

Notare che k-nn fa una approssimazione locale della funzione target per ogni nuovo esempio da classificare, il costo computazionale è tutto contenuto nella fase di predizione. Inoltre è molto costoso a livello computazionale perché per ogni nuovo input bisogna calcolare le distanze dall'esempio a tutti i vettori memorizzati. Il tempo è proporzionale al numero di modelli memorizzati, questo ci fa notare che anche il costo per lo spazio è alto, dato che tutti i dati sono memorizzati.

Quando abbiamo molte variabili in input k-nn spesso fallisce a causa del "curse of dimensionality": quando la dimensione aumenta, il volume dello spazio aumenta in modo tale che i dati disponibili diventano sparsi. Ad esempio la quantità di dati che servono per supportare un risultato spesso cresce esponenzialmente con la dimensione. Troviamo anche il problema denominato "curse of noisy": se il target dipende da poche altre variabili, potremmo trovarci un "modello simile" con la somiglianza dominata dal gran numero di funzioni irrilevanti.

### 9.4 Distance Based methods

Quando utilizzo degli approcci basati su distanza (come k-nn o alcune funzioni kernel) devo assicurarmi che la distanza calcolata sia davvero un fattore discriminante. Stiamo misurando quando una coppia di pattern si assomiglia, inoltre dare una metrica pone un bias rilevante sulla soluzione e la metrica dipende da un dominio (es due stringhe in un linguaggio). Possiamo imparare in qualche modo la metrica? Sì (NN, Learning K...).

## 10 Unsupervised Learning (ripasso)

Nel Unsupervised Learning non abbiamo esempi etichettati, e quindi con un output associato, siamo noi come sistema di apprendimento che dobbiamo trovare correlazioni tra i dati e raggrupparli (clustering). Dobbiamo quindi ripartire i dati, in cluster (sottoinsiemi di dati simili). Il goal che ci poniamo è ripartire in modo ottimo una distribuzione sconosciuta di dati, in uno spazio di dimensione  $x$ , in regioni (approssimando poi usando un cluster). In questo caso  $H$  è un insieme di quantizzatori di vettori  $x \rightarrow c(x)$  cioè dato un vettore in input voglio sapere a quale cluster appartiene. Passiamo da uno spazio continuo ad uno spazio spazio discreto.

Una funzione di Loss comune è

$$L(h(x_i)) = \|x_i - c(x_i)\|^2$$

cioè la distanza tra  $x$  e il centroide. Il valore medio sulla distribuzione degli input è l'errore di quantizzazione.

Unsupervised Learning viene spesso usato in Data Analysis per scoprire caratteristiche comuni tra i dati, viene anche usato per il Preprocessing dei dati da utilizzare poi in seguito per altri approcci di ML. Inoltre dobbiamo notare il fatto che raccogliere dati non etichettati è più "economico", rispetto a trovarne labeled.

### 10.1 K-means

K-means fa parte del Unsupervised Learning. Il K-means è l'algoritmo più semplice e più comunemente usato che utilizza un criterio di errore al quadrato. L'algoritmo K-means è popolare perché è facile da implementare ed è generalmente efficiente, tuttavia, presenta diversi inconvenienti e limitazioni che vedremo dopo...

1. Scegliere k centri di cluster in modo che coincidano con k modelli scelti casualmente (o k punti definiti casualmente all'interno dell'ipervolume contenente il set di pattern)  $c_1 \dots c_k$  con k fissato che sceglio noi.
2. Assegnare ad ogni modello il centro del cluster più vicino (il vincitore) per ogni x calcoliamo

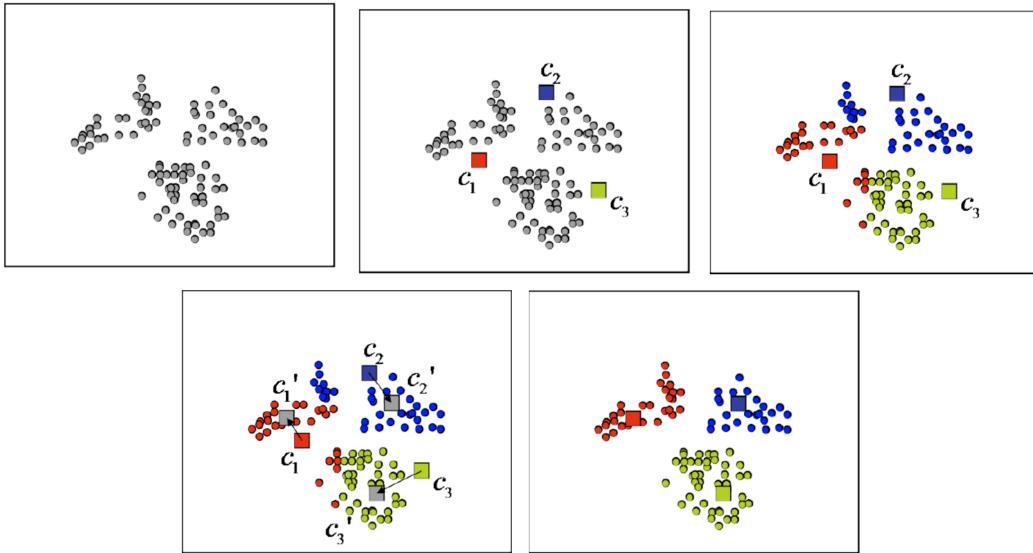
$$i^*(x) = \operatorname{argmin}_i \|x - c_i\|^2 = \sum_{j \rightarrow n} (x_j - c_{ij})^2$$

cioè provo tutti i  $c_i$  e quello che ha distanza minima è il vincitore (mi interessa l'indice del vincitore  $i^*$ ). Adesso x appartiene al cluster  $i^*$

3. Ricalcolare i centri del cluster (centroide) utilizzando le nuove appartenenze al cluster corrente

$$c_i = \frac{1}{|\text{num di membri cluster}_i|} \sum_{x_j \in \text{cluster}_i} x_j$$

4. Se non viene soddisfatto un criterio di convergenza, andare al passaggio 2 (criteri come nessuna (o minima) riassegnazione di schemi a nuovi centri di cluster o una minima diminuzione dell'errore quadratico)



## 10.2 Limitazioni K-means

Il numero di cluster da trovare deve essere fornito (questo porta a fare trial and error per trovare il K che fitta meglio). I minimi locali della Loss rendono il metodo dipendente dall'inizializzazione, si esegue più volte da diverse inizializzazioni casuali (ci sono anche dei metodi che inizializzano con un'euristica). K-means può funzionare bene per cluster compatti, ma non consente di proiettare i dati in uno spazio di dimensione minore.

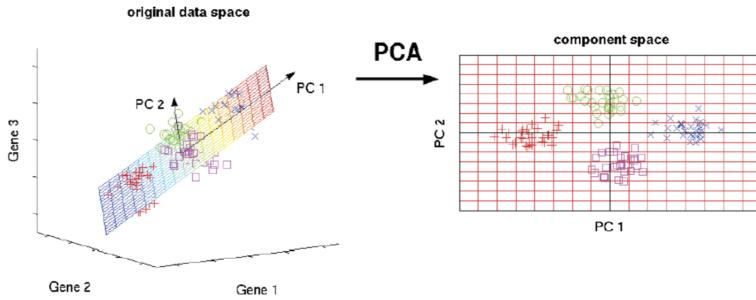
Come viene valutato l'output di un algoritmo di clustering? Che cosa caratterizza un risultato di raggruppamento "buono" e uno "scarso"? Nel clustering esiste ben poco in termini di "standard di riferimento", tranne nei sottodomini specifici in cui conosco la metrica (conosco bene il problema e so riconoscere una classificazione sensata). Misure oggettive (non trattate qui) come ad esempio l'errore di quantizzazione. Noi sappiamo a quali classi appartengono i dati, ma usiamo un algoritmo di clustering, e controlliamo che le classi corrispondano. È un modo sbagliato per valutare la bontà di un algoritmo di clustering!

### 10.3 Preprocessing dei dati

Abbiamo visto che Unsupervised Learning viene utilizzato per il preprocessing dei dati, ma in che modo? Menzioniamo la riduzione della dimensionalità cioè trasformare la dimensione dei dati in una più piccola. (non è importante la quantità, bensì la dimensione dei dati!)

$$\langle x_1, x_2, \dots, x_n \rangle \rightarrow \langle x'_1, x'_2, \dots, x'_m \rangle \text{ con } n > m.$$

Un esempio è il PCA (principal component analysis) dove i nuovi assi sono calcolati nella direzione di massima varianza dei dati



Un altro approccio per il preprocessing dei dati è la feature selection: è una particolare tecnica di riduzione della dimensionalità nella quale scelgo le variabili più significative per il task, anziché trasformarle. In pratica scegliamo un sottoinsieme di tutte le caratteristiche. Un ultimo approccio dal preprocessing che vediamo è Outlier detection: si cercano valori inusuali dei dati che non sono consistenti con gli altri (valori scorretti a causa di misurazioni errate).

## 11 Altri Task del Machine Learning

Altri task che non fanno parte del supervised o unsupervised learning.

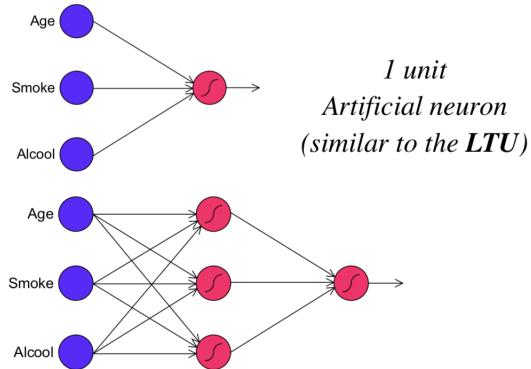
- Reinforcement Learning: si utilizza come metodo di adattamento per sistemi autonomi (in particolare in robotica). L'algoritmo apprende un criterio su come agire data un'osservazione del mondo. Ogni azione ha un certo impatto sull'ambiente e l'ambiente fornisce un feedback che guida l'algoritmo di apprendimento. Invece di avere una supervisione per ogni passaggio, abbiamo informazioni su vittorie / perdite per lo stato finale. Le azioni devono massimizzare la quantità di vittorie ricevute. L'apprendimento decide quali azioni sono state maggiormente responsabili di vittorie / perdite.
- Semi-Supervised Learning: combina esempi etichettati e non etichettati (in genere in numero maggiore) per generare una funzione o un classificatore appropriati.
- Learn to Rank: (utilizzato per i motori di ricerca) quando l'input è un insieme di oggetti e l'output desiderato è una classifica (un ranking) di tali oggetti.
- On-Line Learning: nuovi esempi sono imparati al momento
- Structured domain learning and relational learning: il dominio di input e output può essere strutturato sotto forma di sequenze (segnali, serie temporali, ...) o in modo più complesso: alberi, grafici, reti sociali.

## 12 Altri Modelli del Machine Learning

### 12.1 Reti Neurali

Vengono utilizzate sia nel Supervised che nel Unsupervised Learning. Sono simili alla visione del Linear Threshold Unit, una rete neurale è una rete di modelli non lineari con una capacità di approssimazione universale e capacità predittive. Gli strati interni sono "nascosti", e ogni unità è non lineare, fornendo alla rete neurale la capacità di estrarre (imparando) una nuova rappresentazione dei dati. Questa nuova rappresentazione semplifica il compito di classificazione nell'ultimo livello della rete.

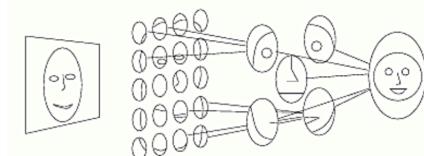
Abbiamo una espansione della base non lineare in  $w$ , e le  $\phi$  sono imparate automaticamente, ma purtroppo questo scaturisce in un problema di ottimizzazione non lineare.



### 12.2 Deep Learning

Il Deep Learning è quel campo di ricerca dell'apprendimento automatico (machine learning) e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso. In altre parole, si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa.

Più tecnicamente, quando si parla di Deep Learning, si fa riferimento a reti neurali multistrato profonde, nel senso che abbiamo molteplici layer di unità di processing non lineari. Indipendentemente se si usa il Supervised o Unsupervised Learning per la rappresentazione di caratteristiche in ogni livello, i vari livelli vanno a formare una gerarchia di caratteristiche / rappresentazioni da basso livello ad alto livello (diversi livelli di astrazione). Imparare automaticamente come rappresentare i dati, e capire come classificarli aumenta il livello di astrazione tra i vari layer. Ad esempio, un'immagine può essere rappresentata in molti modi: come un vettore di pixel o in un modo più astratto come un insieme di bordi, regioni di forma particolare, ecc... Per questo il Deep Learning funziona meglio quando i dati in input hanno una sorta di struttura: spaziale, temporale, linguistica ecc...



I vantaggi del Deep Learning sono svariati, come ad esempio la possibilità di sfruttare la composizionalità della rappresentazione interna: guadagno esponenziale nel potere di rappresentazione. I concetti più semplici vengono rappresentati in uno strato della rete, i quali poi possono essere sfruttati

come dati primitivi dal livello successivo, per rappresentare concetti più complessi. Sono necessari meno esempi per raggiungere una buona capacità di generalizzazione! Le Deep Networks erano difficili da addestrare in passato, ma combinando tecniche per l'addestramento di modelli grandi, HPC (ad esempio GPU) e grandi raccolte di dati da applicazioni reali (ad esempio milioni di immagini), al giorno d'oggi lavorano molto bene conseguendo una rivoluzione nell'approccio AI alle soluzioni del mondo reale.