

APPRENDIMENTO SUPERVISIONATO

Nicola Fanizzi

Ingegneria della Conoscenza

CdL in **Informatica** • *Dipartimento di Informatica*

Università degli studi di Bari Aldo Moro

Apprendimento Automatico e sue Problematiche

Apprendimento in Generale

Problemi di Apprendimento Automatico

Schema di KBS: Apprendimento e Predizione

Tecniche di Apprendimento: Problematiche

Apprendimento Supervisionato

Valutare le Predizioni

Tagli e variabili-Indicatore

Misure dell'Errore di Predizione

Misure per Predizioni Binarie

Tipi di Errore

Matrice di Confusione

Confronto di Modelli

Stime Puntuali senza Feature di Input ↵

Modelli-base per l'Apprendimento Supervisionato

Alberi di Decisione

Regressione e Classificazione Lineari

Discesa di Gradiente (DG)

Classificazione: Modelli con Funzioni Lineari

Appiattite

Regressione Logistica

Separabilità Lineare

Sovradattamento

Pseudoconteggi

Regolarizzazione

Cross Validation

k-Fold Cross Validation

Leave-One-Out Cross Validation

Reti Neurali e Apprendimento Profondo

Reti Neurali Artificiali

Architettura

Back-Propagation

Modularizzazione

Estensioni dei Modelli Lineari

Support Vector Machine (SVM)

Modelli Compositi

Ensemble Learning

Bagging: Random Forest

Boosting

Stacking

Case-Based Reasoning

k-Nearest Neighbors

Apprendimento come Ricerca: Raffinamento di Ipotesi ↵

APPRENDIMENTO AUTOMATICO E SUE PROBLEMATICHE

Apprendimento in Generale

Apprendimento capacità di sfruttare l'*esperienza* per migliorare il *comportamento* nello svolgimento di specifici compiti

- *estensione* delle abilità / comportamenti possibili:
 - fare di più
- miglioramento dell'*accuratezza* nei compiti svolti:
 - far meglio
- miglioramento dell'*efficienza*:
 - fare prima

Focus sull'**apprendimento supervisionato** di modelli predittivi:

- dato un insieme di **esempi**, coppie input+output,
predire l'output per nuovi casi di cui si conosce solo l'input

Approcci

- scegliere una **singola ipotesi** di modello che si adatti bene agli esempi dati
 - predizioni **direttamente** a partire dagli esempi
 - predizioni selezionando un sottospazio di **ipotesi coerenti** con gli esempi
 - predizioni in base a distribuzioni di **probabilità** delle ipotesi **condizionate** dagli esempi osservati
- nel seguito: diversi modelli, rappresentazioni e forme di apprendimento*

PROBLEMI DI APPRENDIMENTO AUTOMATICO

Task *comportamento* o *compito* nel quale migliorare

Dati *esperienze* usate per migliorare le prestazioni nel task:

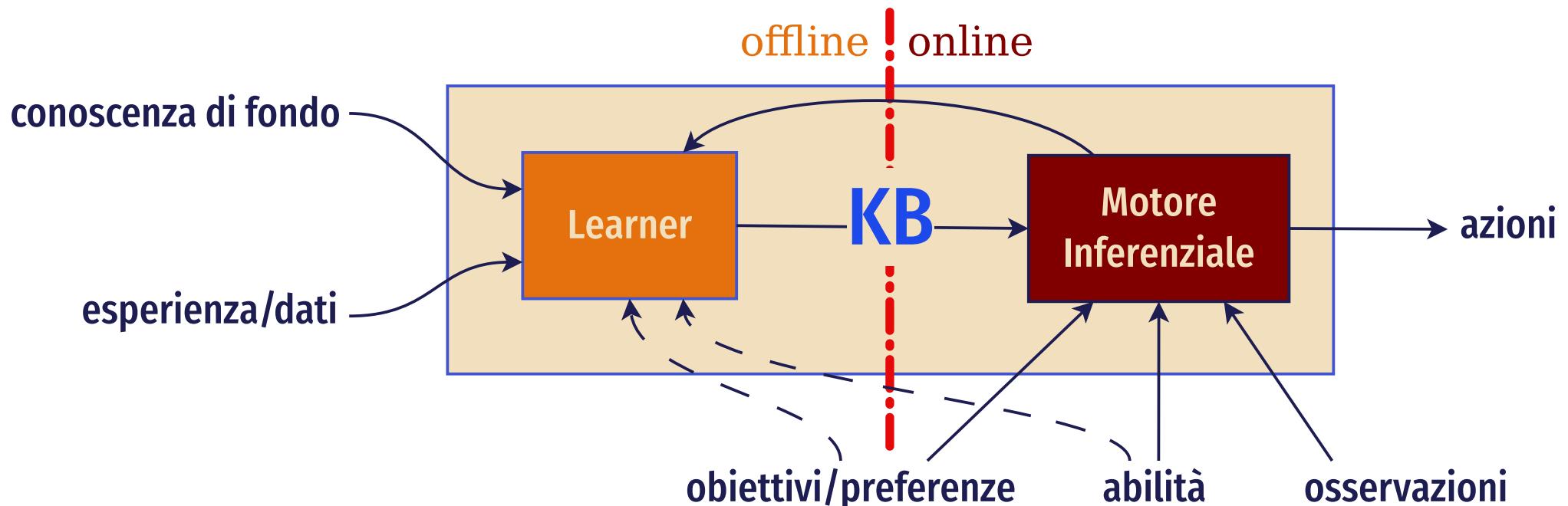
- sequenza di *esempi*, di solito

Misura del miglioramento – *come* / *cosa* misurare:

- incremento di abilità, efficacia, efficienza

SCHEMA DI KBS: APPRENDIMENTO E PREDIZIONE

- si considerano:
 - *conoscenza di fondo (background knowledge, BK)*
 - *dati*: esperienza pregressa, esempi: coppie *input* → *output*
- per creare una *rappresentazione interna*, (parte della) KB da usare per future azioni/decisioni: *predizioni*



TECNICHE DI APPRENDIMENTO: PROBLEMATICHE

- **Task** – compito per cui acquisire dati / esperienza
 - **apprendimento supervisionato**: dato un insieme di esempi di training descritti in termini di **feature** (caratteristiche, attributi) di **input** e **target** (obiettivo), imparare a predire (online) il valore ignoto target per nuove osservazioni
 - **classificazione**: target discreto vs. **regressione**: target continuo
 - altri task/obiettivi:
 - **apprendimento non-supervisionato**: regolarità / pattern, senza target esplicativi
 - **concept / relational learning** : modelli per rappresentazioni più espressive
 - e.g. concetti, relazioni, programmi logici (ILP)
 - **reinforcement learning**: comportamenti attraverso *ricompense e punizioni*
 - **analytic learning**: per ragionare / prendere decisioni più velocemente
 - **ranking**: imparare a ordinare → rec. sys

- **Feedback** – dipende dal task di apprendimento
 - supervisionato: **esempi di training** forniti da un esperto/addestratore, con il corretto valore-target per ognuno
 - non-supervisionato: **osservazioni** delle sole feature di input, ~~output~~
 - si devono scoprire categorie / regolarità nei dati
 - per rinforzo: **ricompense e punizioni**, il valore delle azioni/decisioni
 - problema di **credit assignment**: determinare le azioni responsabili delle ricompense e delle punizioni
- **Offline/Online** – acquisizione esempi
 - **offline**: tutti disponibili prima dell'azione/decisione
 - **online**: nel tempo, nel corso del processo di learning
 - rappresentazione degli esempi pregressi prima di averli visti tutti
 - nuovi esempi → aggiornamento della rappresentazione
 - tipicamente non si possono osservare *tutti* i potenziali esempi
 - **active learning**: il sistema ragiona e decide gli esempi da acquisire

- **Rappresentazione** – *codifica interna* dell'esperienza: ipotesi / modello predittivo
 - forma **grezza**, le esperienze stesse, gli esempi, oppure forma **compatta**, più frequente, generalizzazioni dei dati
 - apprendimento come problema di **induzione**
 - dagli esempi particolari a conclusioni generali
 - opposto della deduzione e diverso dall'abduzione (non spiegazioni specifiche)
 - **principi-guida** per la scelta:
 - rappresentazione più ricca → più utile a esprimere ipotesi generali ma più difficile da apprendere
 - servono più dati
 - per scegliere fra le più numerose ipotesi coerenti con i dati disponibili
 - compromessi: metodi di learning → rappresentazione delle ipotesi
 - ad es., alberi di decisione, reti neurali, ecc.

- **Misura delle prestazioni** – miglioramento dei modelli misurato su *nuovi* dati
 - **valutazione**: tipicamente si dividono gli esempi in *training* e *test*
 - si costruisce una rappresentazione del *modello/ipotesi* sugli esempi di training
 - si misura l'*accuratezza predittiva* su esempi di test, non considerati nel training
 - misura *approssimata* delle prestazioni
da ripetere al variare delle suddivisioni del campione di dati
 - caso supervisionato: ipotesi/classificatori, indotti da *campioni* della popolazione
 - devono *generalizzare* il training set, senza farsi fuorviare da sue specificità
 - altrimenti risulterebbero scarsamente *predittivi* rispetto a nuovi dati
 - ad es. *problema di classificazione binario* (appartenenza a una classe)
 - h_P : predice tutti gli esempi come positivi, tranne i soli negativi del training set
 - h_N : predice tutti gli esempi come negativi, tranne i soli positivi del training set
 - ipotesi *estreme*: per entrambe **100% di accuratezza** sul training
ma in *totale disaccordo* sul resto → le ipotesi vanno valutate su altri dati

- **Bias** – tendenza a preferire certe ipotesi rispetto ad altre
 - senza bias, come stabilire che un'ipotesi sia migliore di un'altra?
 - h_N o h_P entrambe accurate al 100% sui dati di training: come scegliere?
 - **fattore esterno**: anche a prescindere dai dati, per predizioni su esempi non visti
 - h_P e h_N saranno in disaccordo su tutti gli esempi futuri
 - **scelta del bias**: problema empirico da risolvere attraverso la pratica
 - tipicamente, si basa sulla *complessità* delle ipotesi *(rasoio di Occam)*

- **Ricerca** – scelti rappresentazione e bias:

apprendimento = ***problema di ricerca***

- si attraversa lo ***spazio delle ipotesi*** (rappresentazioni possibili del modello) per trovare quella che meglio si adatta agli esempi, dato il bias
 - tipicamente spazi molto estesi, spesso infiniti
 - ~~ricerca sistematica~~ → ***ricerca locale***
- la definizione dell'***algoritmo*** richiede:
 - spazio di ricerca
 - funzione di valutazione
 - metodo di ricerca

- **Rumore – dati reali *imperfetti***
 - **rumore**: feature osservate inadeguate a predire la classificazione
 - **dati mancanti**: per certi esempi valori di alcune feature non osservati
 - **errori**: valori errati di alcune feature
- **Ordine naturale del dominio delle feature (es. nello spazio, nel tempo)**
 - **interpolazione**: predizioni per casi *compresi tra* i dati osservati
 - **estrapolazione**: predizioni per casi che *vanno oltre* gli esempi visti
 - meno accurata, in genere: “*più facile predire il passato che il futuro*”
 - determinare il valore incognito di una feature, in base a valori già osservati prima e dopo, molto più facile che predire valori futuri
 - **NB** parametri di ipotesi ottimizzati ai fini dell'*interpolazione* possono rivelarsi pessimi per l'*estrapolazione*

APPRENDIMENTO SUPERVISIONATO

Problema Supervisionato

Feature (in It., caratteristica o attributo): funzione F sugli *esempi*

- $F(e)$: **valore** di F per l'esempio e
- **dominio** della feature: insieme dei valori che può assumere
 - ossia il codominio della funzione F

Problema

- **dati:**
 - **feature di input:** X_1, \dots, X_n
 - **feature-obiettivo (o target):** Y_1, \dots, Y_k spesso $k = 1$
 - insieme di esempi suddiviso in
 - **training** set, con valori disponibili per tutte le feature
 - **test** set, con valori solo per le feature di input
- **imparare** sulla base del training set un'ipotesi, def. (parametrica) di una funzione, atta a predire i valori delle Y_j degli esempi di test (e di altri disponibili in futuro)

Esempio – Problema di **classificazione** di post in *thread* d'un sito di discussioni

- Feature di **input** (binarie):
 - *Author*: notorietà dell'autore, con dominio $\{known, unknown\}$,
 - *Thread*: novità nel thread, con dominio $\{new, followup\}$,
 - *Length*: lunghezza dl post, con dominio $\{long, short\}$,
 - *Where_read*: luogo di lettura, con dominio $\{home, work\}$
- Feature **target**: *User_action* con dominio $\{reads, skips\}$
- Esempi di **training**: e_1, \dots, e_{18} e **test**: e_{19}, e_{20}
 - e.g., nel dataset:
 - $Author(e_{11}) = unknown$,
 - $Thread(e_{11}) = followup$
 - $User_Action(e_{11}) = skips$,
 - ...

- Tabella preferenze utente – esempi ottenuti dalle *decisioni* degli utenti:
 - lettura/salto di un dato post

<i>E</i>	<i>Author</i>	<i>Thread</i>	<i>Length</i>	<i>Where_read</i>	<i>User_action</i>
<i>e</i> ₁	known	new	long	home	skips
<i>e</i> ₂	unknown	new	short	work	reads
<i>e</i> ₃	unknown	followup	long	work	skips
<i>e</i> ₄	known	followup	long	home	skips
<i>e</i> ₅	known	new	short	home	reads
<i>e</i> ₆	known	followup	long	work	skips
<i>e</i> ₇	unknown	followup	short	work	skips
<i>e</i> ₈	unknown	new	short	work	reads
<i>e</i> ₉	known	followup	long	home	skips
<i>e</i> ₁₀	known	new	long	work	skips
<i>e</i> ₁₁	unknown	followup	short	home	skips
<i>e</i> ₁₂	known	new	long	work	skips
<i>e</i> ₁₃	known	followup	short	home	reads
<i>e</i> ₁₄	known	new	short	work	reads
<i>e</i> ₁₅	known	new	short	home	reads
<i>e</i> ₁₆	known	followup	short	work	reads
<i>e</i> ₁₇	known	new	short	home	reads
<i>e</i> ₁₈	unknown	new	short	work	reads
<i>e</i> ₁₉	unknown	new	long	work	?
<i>e</i> ₂₀	unknown	followup	short	home	?

Esempio – Problema di *regressione*: predire il valore di Y a valori reali

es.	X	Y
e_1	0.7	1.7
e_2	1.1	2.4
e_3	1.3	2.5
e_4	1.9	1.7
e_5	2.6	2.1
e_6	3.1	2.3
e_7	3.9	7.0
e_8	2.9	?
e_9	5.0	?

- predire il valore di Y per e_8 : problema di *interpolazione*
 - $X(e_8)$ ricade tra quelli degli esempi di training
- predire il valore di Y per e_9 : problema di *estrapolazione*
 - $X(e_9)$ fuori dell'intervallo di valori degli esempi di training

Valutare le Predizioni

$\hat{Y}(e)$ stima puntuale di Y su e : predizione del valore $Y(e)$

Errore di $\hat{Y}(e)$: misura della *distanza* di $\hat{Y}(e)$ da $Y(e)$

- **regressione**: $\hat{Y}(e)$ e $Y(e)$ numeri reali confrontabili aritmeticamente
- **classificazione**: Y discreta, varie alternative possibili:
 - Y binaria mappata su $\{0, 1\}$ → stima analoga oppure numero reale in $[0, 1]$
 - valori confrontabili numericamente
 - in alternativa anche $\{-1, 1\}$ o zero / non-zero
 - Y cardinale con valori mappati sui reali
 - **differenza** appropriata per valori totalmente ordinati:
 - stime e valori reali confrontabili su tale scala
 - Y ordinale: ordine totale ma stime e valori reali non confrontabili
 - es. dato il dominio $\{small, medium, large\}$,
predizione $\hat{Y}(e) \in \{small, large\}$ molto diversa da $\hat{Y}(e) = medium$

TAGLI E VARIABILI-INDICATORE

- se Y **totalmente ordinata**, dato un suo valore v , si può definire la feature binaria $Y \leq v$ mediante **cut (taglio)** con valori (0 o 1) determinati in base a $Y(e) \leq v$
 - v noto a priori o ricavato dai dati
 - altre relazioni possibili, e.g. <
 - combinando più tagli → feature con dominio fatto di **intervalli**

NB taglio su max/min inutile
- se Y ha dominio $\{v_1, \dots, v_k\}$, $k > 2$,
si possono avere predizioni **separate** per ciascun v_i
 - usando **variabili-indicatore** binarie Y_i associate a ogni v_i , dove
 - $Y_i(e) = \begin{cases} 1 & Y(e) = v_i \\ 0 & \text{altrimenti} \end{cases}$
 - solo una tra $Y_1(e), \dots, Y_k(e)$ vale 1, le altre 0
 - **predizione:** k valori in $[0, 1]$, uno per ciascun Y_i

Esempio – Imparare le preferenze degli utenti sulla lunghezza delle vacanze (supponendo si vada da 1 a 6 giorni)

1. *Y* a *valori reali*: numero giorni di vacanza

<i>E</i>	<i>Y</i>
<i>e</i> ₁	1
<i>e</i> ₂	6
<i>e</i> ₃	6
<i>e</i> ₄	2
<i>e</i> ₅	1

- predizione per un nuovo esempio *e*:

$$\hat{Y}(e) = 3.2$$

2. con **variabili-indicatore**, Y_1, \dots, Y_6

- Y_i rappresenta la proposizione: “si preferisce una vacanza di i giorni”

E	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
e_1	1	0	0	0	0	0
e_2	0	0	0	0	0	1
e_3	0	0	0	0	0	1
e_4	0	1	0	0	0	0
e_5	1	0	0	0	0	0

- si dovrebbe predire il valore di ogni Y_i per ogni e
 - ad es. $\hat{Y}(e) = \langle 0.5, 0.3, 0.1, 0.14, 0.1, 0.5 \rangle$
 - si preferisce di più un periodo di uno o sei giorni (e meno le altre soluzioni)

3. *tagli binari* $Y \leq i$ per i diversi valori di i :

E	$Y \leq 1$	$Y \leq 2$	$Y \leq 3$	$Y \leq 4$	$Y \leq 5$
e_1	1	1	1	1	1
e_2	0	0	0	0	0
e_3	0	0	0	0	0
e_4	0	1	1	1	1
e_5	1	1	1	1	1

- per ogni i si predice un valore associato a $Y \leq i$:
 - ad es. $Y \leq 1$ con valore 0.4 e $Y \leq i$ con 0.6 per gli altri valori di i
 - **NB** non razionale predire, ad es., $Y \leq 2$ ma non $Y \leq 4$: uno implica l'altro

MISURE DELL'ERRORE DI PREDIZIONE

Nel seguito:

- E insieme di esempi
- T insieme di feature obiettivo (target)
- dati $Y \in T$ e $e \in E$:

$Y(e)$ valore reale e $\hat{Y}(e)$ valore predetto

- **errore 0/1** su E : somma del *numero* di predizioni errate sugli esempi e per target

$$\sum_{e \in E} \sum_{Y \in \mathbf{T}} Y(e) \neq \hat{Y}(e)$$

dove $Y(e) \neq \hat{Y}(e)$ vale 0 se falso e 1 se vero, indicato anche con¹ $\delta(\cdot, \cdot)$

- **NB** non tiene conto di *quanto* siano errate

- **errore assoluto** su E : somma delle *differenze assolute* tra valori effettivi e loro stime su tutti gli esempi e target

$$\sum_{e \in E} \sum_{Y \in \mathbf{T}} |Y(e) - \hat{Y}(e)|$$

- sempre non negativo: *nullo* → predizioni corrette
- *dipende dalla precisione* delle predizioni

- somma dei quadrati degli errori su E :

$$\sum_{e \in E} \sum_{Y \in \mathbf{T}} (Y(e) - \hat{Y}(e))^2$$

- grandi errori molto peggiori rispetto ai piccoli
 - ad es. un errore di 2 vale quanto 4 errori di 1
e un errore di 10 vale quanto 100 errori di 1
- equivalentemente si può adottare la **radice dell'errore quadratico medio** (*root-mean-square error*, RMSE) ossia dell'MSE, mediato dividendo per $|E|$
 - stessi punti di minimo
- **errore nel caso pessimo** su E : **massima differenza** in termini assoluti tra stime e valori effettivi su tutti al variare di esempi e target

$$\max_{e \in E} \max_{Y \in \mathbf{T}} |Y(e) - \hat{Y}(e)|$$

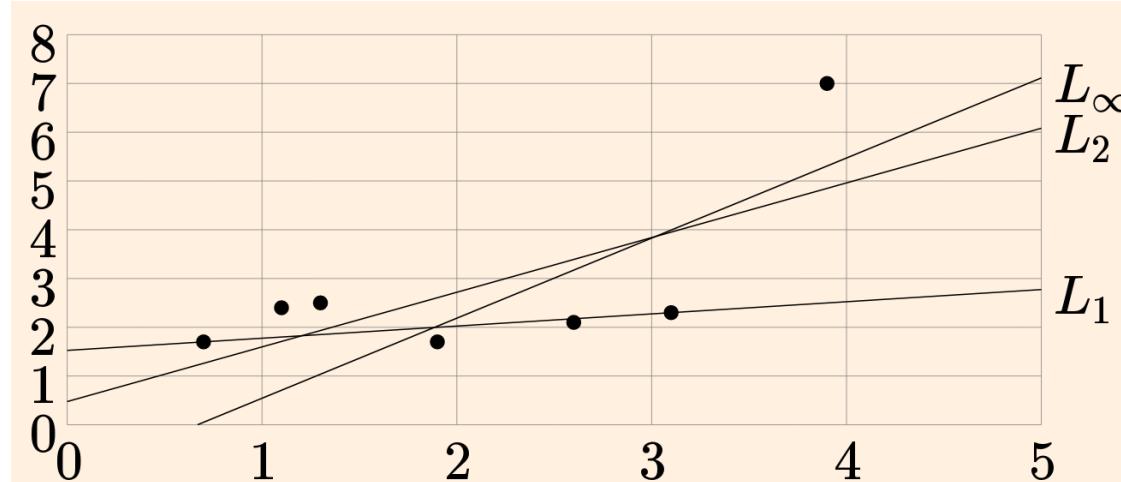
- valutazione basata su quanto possa sbagliare un modello

NORME

Errori spesso definiti in termini di norme² delle differenze (vettoriali):

- errore 0/1 → *errore L_0*
 - non esattamente una norma
- errore assoluto → *errore L_1*
- errore somma dei quadrati → *errore L_2*
 - scritta anche L_2^2 perché la L_2 considera la somma dei quadrati sotto radice q.
- errore nel caso pessimo → *errore L_∞*

Esempio – Problema di regressione precedente



- rette che predicono Y minimizzando l'errore L_1 , L_2 e L_∞
 - ogni retta passante per 2 punti minimizza l'errore L_0
 - tutte le triple *non co-lineari*
 - per $X = 1.1$ (valore effettivo $Y = 2.4$):
 - dalle rette L_1 e L_2 predizioni *simili*: risp. 1.805 e 1.709
 - L_∞ predice 0.7
 - interpolando in $[1, 3]$ danno predizioni a distanza 1.5
 - estrapolando, *divergono*:
 - ad es. per $X = 5$, L_1 e L_∞ forniscono predizioni molto differenti

Outlier esempio che non segue il *pattern* degli altri esempi

- Differenza tra rette più evidente nel trattamento degli outlier, come (3.9, 7)
 - predizione con L_∞ dipende solo da (1.1, 2.4), (3.1, 2.3) e (3.9, 7)
 - stesso errore di caso pessimo
 - altri punti ininfluenti
 - predizione con L_1 non cambia in base al valore effettivo di Y per gli esempi, se i punti restano al di sopra e al di sotto, risp.
 - ad es., stessa predizione anche per il punto (3.9, 107) anziché (3.9, 7)
 - predizione con L_2 sensibile a tutti i dati
 - cambiando la Y di qualche esempio, cambia la retta
 - cambiamenti negli outlier hanno maggiori effetti sulla retta rispetto ai punti ad essa vicini

MISURE PER PREDIZIONI BINARIE

Caso speciale: $Y(e) \in \{0, 1\}$, predizione $\hat{Y}(e) = [0, 1]$

- **verosimiglianza (*likelihood*)** dei dati E
interpretando $\hat{Y}(e)$ come probabilità, con predizioni indipendenti sugli esempi

$$L = \prod_{e \in E} \prod_{Y \in \mathbf{T}} \hat{Y}(e)^{Y(e)} (1 - \hat{Y}(e))^{(1-Y(e))}$$

- esponente 1 o 0 → fattore semplificato: $\hat{Y}(e)$ o $(1 - \hat{Y}(e))$
- massimizzandola → stima di **massima verosimiglianza** (*maximum likelihood*, ML)
- **log-likelihood** logaritmo della verosimiglianza:

$$LL = \sum_{e \in E} \sum_{Y \in \mathbf{T}} [Y(e) \log(\hat{Y}(e)) + (1 - Y(e)) \log(1 - \hat{Y}(e))]$$

- da massimizzare (come la precedente)
- da minimizzare: **log-loss** $-LL/|E|$

(cfr. *entropia*)

Esempio – Si consideri il dataset delle vacanze visto prima

- considerando un semplice **classificatore costante** (stessa predizione per tutti gli esempi)

1. con unica Y , predizione che minimizza:

- L_1 : 2 (errore: 10)
- L_2 : 3.2
- L_∞ : 3.5

2. con $Y_1 \dots Y_6$, predizione che minimizza:

- L_1 : 0 per ogni Y_i
- L_2 : $Y_1 = 0.4, Y_2 = 0.1, Y_3 = 0, Y_4 = 0, Y_5 = 0$ e $Y_6 = 0.4$
 - vale anche per la likelihood
- L_∞ : $Y_1 = Y_2 = Y_6 = 0.5$ e $Y_3 = Y_4 = Y_5 = 0$

Predizione da preferire in base a:

- **rappresentazione** del classificatore
- **misura di valutazione**

Codifica Binaria

cfr. specchietto [1] e [8, 9]

Bit: quantità d'informazione utile codificare 2 simboli distinti

- con **1 bit** (lunghezza) si formano **2^l** codici binari
 - che distinguono altrettanti simboli
- per distinguere **n** simboli, lunghezza codici: **$\log_2 n$ bit** problema inverso
 - stessa lunghezza **n** assumendo simboli **equiprobabili**: $\forall x \ P(x) = \frac{1}{n}$ (costante)
- in generale, nota la **distribuzione P** dei simboli, per codificare **x** , lung. ottimale:

$$\log_2 \frac{1}{P(x)} = -\log_2 P(x) \text{ bit}$$

- simboli più probabili (frequenti) → codici più corti

- **Entropia (contenuto informativo) di $P(x)$:**
per trasmettere una *sequenza codificata* di simboli, ognuno richiede in media

$$H_{P(x)} = \sum_x P(x) \cdot (-\log_2 P(x)) \text{ bit}$$

- *Entropia condizionata* di $P(x|e)$, data l'evidenza e :

$$H_{P(x|e)} = \sum_x -P(x|e) \cdot \log_2 P(x|e)$$

- *Informazione attesa* di un *test a 2 vie*, condizione α (ad es. $F(\cdot) = v$) entropia condizionata mediando sui 2 esiti osservabili, α e $\neg\alpha$:

$$\sum_{e \in \{\alpha, \neg\alpha\}} P(e) \cdot H_{P(x|e)}$$

- **Information Gain (IG)** di un test: guadagno di dovuto al test come differenza tra entropia iniziale e informazione attesa dopo il test

Alcuni tipi di errore possono essere peggiori di altri nelle conseguenze

Esempio – in medicina non diagnosticare una malattia a un paziente, che non si curerà, può essere più grave rispetto a predire una malattia che non ha, consigliando quindi ulteriori accertamenti

Predizione come *azione*:

- scelta migliore in termini di *costi* associati agli errori
- decisione legata anche a fattori di *incertezza* discussi più avanti

MATRICE DI CONFUSIONE

Caso base con Y booleana

- valutazione della predizione indipendente dalla decisione:

	ap actual positive	an actual negative
pp predict positive	tp true positive	fp <i>false positive</i>
pn predict negative	fn <i>false negative</i>	tn true negative

- *Errori*
 - **falso positivo** o errore di *I tipo*: predizione positiva sbagliata
 - *true* invece dell'effettivo valore *false*
 - **falso negativo** o errore di *II tipo*: predizione negativa sbagliata
 - *false* invece dell'effettivo *true*

Misure per un modello predittivo **binario**:

- **precisione (*precision*):**
proporzione di veri positivi (*tp*) rispetto ai casi predetti come positivi (*pp*)

$$\frac{tp}{tp + fp}$$

- **richiamo (*recall* o *true-positive rate*, TPR):**
proporzione di veri positivi (*tp*) rispetto a tutti i positivi effettivi (*ap*)

$$\frac{tp}{tp + fn}$$

- analogamente ***false-positive rate* (FPR):**
proporzione di falsi positivi (*fp*) su tutti i negativi (*an*)

$$\frac{fp}{fp + tn}$$

SCELTA DEL MODELLO PREDITTIVO

Occorrerebbe massimizzare precisione e richiamo e minimizzare FPR:
nella pratica obiettivi incompatibili

- si può massimizzare la precisione e minimizzare FPR:
facendo predizioni positive solo se si è sicuri
 - ma ciò abbassa il richiamo
- per massimizzare il richiamo: ammettere previsioni più azzardate
 - abbassa la precisione e incrementa FPR

CONFRONTO DI MODELLI

Indipendentemente dai costi, si possono considerare:

- **spazio ROC** (*receiver operating characteristic*)
 - grafico del FPR contro il TPR
- **spazio precision-recall**
 - grafico della precisione contro il richiamo

ogni modello produce un punto nello spazio

Esempio – caso di 100 esempi positivi (*ap*) e 1000 esempi negativi (*an*)

matrici di confusione di diversi modelli

a	ap	an
pp	70	150
pn	30	850

b	ap	an
pp	50	25
pn	50	975

c	ap	an
pp	98	200
pn	2	800

d	ap	an
pp	90	500
pn	10	500

e	ap	an
pp	100	1000
pn	0	0

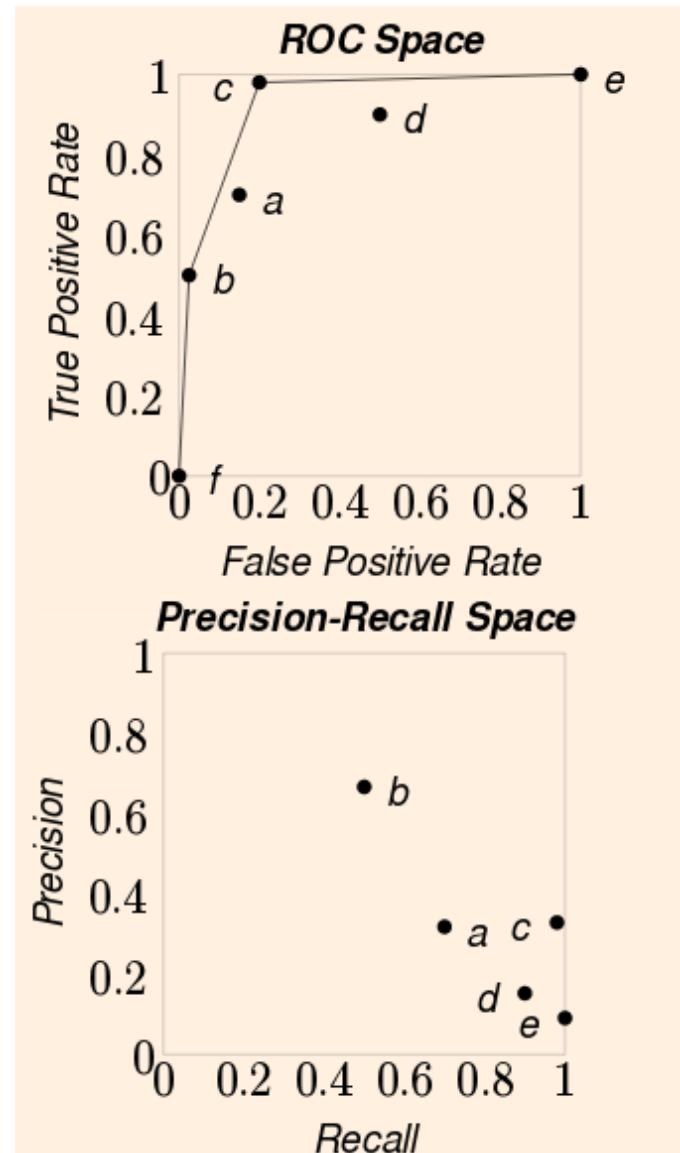
f	ap	an
pp	0	0
pn	100	1000

Confronto tra a e c

- **a**
 - **richiamo:** 0.7
 - **FPR:** 0.15
 - **precisione:** $70/220 \approx 0.318$
- **c**
 - **richiamo:** 0.98
 - **FPR:** 0.2
 - **precisione:** $98/298 \approx 0.329$
- quindi **c** migliore di **a** in termini di precisione e richiamo
ma peggiore in termini di FPR
 - se i falsi positivi fossero considerati più importanti dei falsi negativi,
a sarebbe preferibile rispetto a **c**
 - si vede nello spazio ROC ma non in quello P-R

Spazio ROC

- peggiori modello più in basso e più a destra
 - e.g. **d** è peggiore di **c**:
 - non sarebbe logico preferire **d** a **c**
- ogni modello al di sotto dell'*inviluppo* di modelli limitato dalla spezzata è *dominato* dagli altri
 - ad es. **a**, sebbene non superato da **b** o **c**, sarebbe dominato da un *modello random* che adotti in metà dei casi la predizione di **b** e nell'altra quella di **c**
 - approssimativamente: $fp = 112.5$ e $fn = 26$



Caso semplice: predizione basata solo sulla feature-obiettivo Y (nel training)

- senza usare le feature di input X_i
 - si predice *uno stesso valore* $v = \hat{Y}(e) \forall e \in E$ per tutti gli esempi
 - v dipende dal *tipo di errore* da minimizzare modello **costante**

Si dimostra:

Proposizione – Sia V lista di valori di $Y(e)$ per ogni $e \in E$:

1. errore 0/1 $\sum_{e \in E} Y(e) \neq v$ minimizzato da una **moda**³ di V
2. errore della somma di quadrati $\sum_{e \in E} (Y(e) - v)^2$ minimizzato dalla **media** di V
3. errore assoluto $\sum_{e \in E} |Y(e) - v|$ minimizzato da una **mediana**⁴ di V
4. errore del caso pessimo $\max_{e \in E} |Y(e) - v|$ minimizzato da $\frac{\max(V) + \min(V)}{2}$

Caso – Y binaria

- dati di training riassumibili dai soli n_0 e n_1 (frequenze dei valori di Y)
- predizione **ottimale** p dipendente dal **criterio di ottimalità** (1-4)
 - calcolabile e ottimizzabile **analiticamente** sugli esempi di training

Misure	Misura di p sul training set	p ottimale sul training set
0/1	$\begin{cases} n_1 & \text{se } p = 1 \\ n_0 & \text{se } p = 0 \end{cases}$	$\begin{cases} 0 & \text{se } n_0 > n_1 \\ 1 & \text{altrimenti} \end{cases}$
assoluto	$n_0 p + n_1 (1 - p)$	$\begin{cases} 0 & \text{se } n_0 > n_1 \\ 1 & \text{altrimenti} \end{cases}$
somma dei quadrati	$n_0 p^2 + n_1 (1 - p)^2$	$\frac{n_1}{n_0 + n_1}$
caso pessimo	$\begin{cases} p & \text{se } n_1 = 0 \\ 1 - p & \text{se } n_0 = 0 \\ \max(p, 1 - p) & \text{altrimenti} \end{cases}$	$\begin{cases} 0 & \text{se } n_1 = 0 \\ 1 & \text{se } n_0 = 0 \\ 0.5 & \text{altrimenti} \end{cases}$
likelihood	$p^{n_1} (1 - p)^{n_0}$	$\frac{n_1}{n_0 + n_1}$
log-likelihood	$n_1 \log p + n_0 \log(1 - p)$	$\frac{n_1}{n_0 + n_1}$

Predizione ottimale (caso di Y binaria)

- **dati di training:**
 - per l'errore assoluto: *mediana*
 - che in tal caso è anche la moda
 - errore lineare in $p \rightarrow$ minimo in uno degli estremi
 - per gli altri criteri **frequenza empirica**: proporzione di **1** nel training set, i.e. $\frac{n_1}{n_0+n_1}$
 - interpretabile come predizione di una probabilità
(cfr. in seguito: *stima di massima verosimiglianza*)
 - **dati di test:**
 - la frequenza e. non massimizza necessariamente verosimiglianza o entropia
 - caso limite: $n_0 = 0$ (risp. $n_1 = 0$) $\rightarrow p = 1$ (risp. $p = 0$) per tutti gli esempi
 - con un esempio di test con l'altra classe, likelihood *nulla* ed entropia *infinita*
- caso di **sovradattamento!**

MODELLI-BASE PER L'APPRENDIMENTO SUPERVISIONATO

Modelli di Predizione

Modello da apprendere: funzioni

- ***schema-tipo*** di algoritmo supervisionato *classificatori/regressori*
 - dato un insieme di esempi descritti in termini degli insiemi di feature X_s e Y_s
restituire la rappresentazione compatta di funzioni
utile a future predizioni dei valori delle feature target di esempi
 - per cui siano disponibili i valori delle feature di input
- ***schemi alternativi:***
 - non costruiscono un modello,
fanno predizioni direttamente sulla base degli esempi
 - ad es. CASE-BASED REASONING, K-NN

Differenze fra algoritmi: in base alla rappresentazione delle funzioni

Alberi di Decisione

Si assumerà una sola Y discreta: la **classificazione**

- i cui valori / elementi sono detti **classi**

Albero di decisione (o **di classificazione**) binario costituito da:

- nodi **interni** (non-foglia)
 - etichettati con **condizioni**, test booleani⁵ sui valori delle feature degli esempi
 - con due **figli**, radici di sotto-alberi, etichettati con **true** e **false**, risp.
- nodi-**foglia**
 - etichettati con una stima puntuale della classe
 - analogo a una funzione definita **per parti**
ovvero a un programma con strutture condizionali annidate

CLASSIFICAZIONE

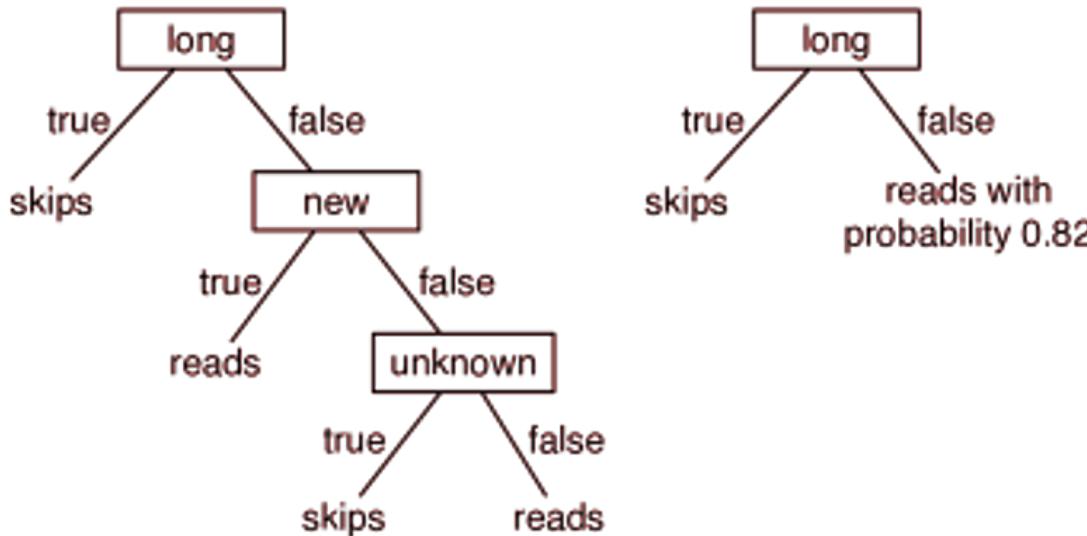
Dato un albero di decisione e un esempio con i valori per le Xs :

- partendo dalla radice, ogni condizione incontrata viene valutata e si segue l'arco corrispondente al risultato
- raggiunta una foglia, si assegna la classe Y corrispondente

Albero come **programma** per determinare una predizione / classificazione:

- **foglie: classe** predetta (senza condizioni)
- **nodi interni:** if $c(e)$ then $t_1(e)$ else $t_0(e)$
 - t_1 e t_0 sotto-alberi
 - t_1 sul ramo true; t_0 sul ramo false

Esempio – Due alberi di decisione per il dataset precedente



◀ definisce la funzione di classificazione:

```
define UserAction(e):  
    if long(e) return skips  
    else  
        if new(e) return reads  
        else  
            if unknown(e) return skips  
            else return reads
```

► consente predizioni probabilistiche:

- ad es. se il post non è *long*,
predice *reads* con probabilità 0.82
 - quindi *skips* avrebbe probabilità 0.18

APPRENDIMENTO DI ALBERI DI DECISIONE

Dati gli esempi di training

- **QUALE** albero generare?
 - rappresentabile *qualsiasi funzione* con X_i discrete
 - *bias* insito nel criterio di preferenza tra i diversi alberi
 - alberi più piccoli coerenti con i dati: minima profondità o con meno nodi
- **COME** generarlo?
 - *ricerca* a partire dal più piccolo che si adatti ai dati
 - !! spazio enorme
 - *ricerca greedy*, minimizzando l'errore

L'algoritmo segue (top-down) la definizione **ricorsiva** di albero:

- **caso base:** secondo il criterio di stop,
si determina una stima da installare nel nodo-foglia
 - chiamata `stima_puntuale(Y, e)`: determina la classe (di maggioranza)
per ogni esempio e che raggiunga il nodo-foglia
- **caso ricorsivo:** si sceglie una condizione c
 - esempi e da partizionare in base al test $c(e)$
 - un sotto-albero per partizione

procedure DT_Learner(C_s, Y, E_s)

Input

C_s : insieme delle possibili condizioni

Y : feature-obiettivo

E_s : insieme di esempi di training

Output

funzione per predire il valore di Y per un esempio

if criterio di stop vero **then**

$v \leftarrow$ stima_puntuale(Y, E_s)

define $T(e) = v$

else

Scegliere una condizione $c \in C_s$

$true_examples \leftarrow \{e \in E_s \mid c(e)\}$

$t_1 \leftarrow$ DT_Learner($C_s \setminus \{c\}, Y, true_examples$)

$false_examples \leftarrow \{e \in E_s \mid \neg c(e)\}$

$t_0 \leftarrow$ DT_Learner($C_s \setminus \{c\}, Y, false_examples$)

define $T(e) =$ if $c(e)$ then $t_1(e)$ else $t_0(e)$

return T

Esempio – Invocando l'algoritmo sui dati visti in precedenza:

- $\text{DT_Learner}(\{\text{Author}, \text{Thread}, \text{Length}, \text{Where_read}\}, \text{User_action}, \{e_1, e_2, \dots, e_{18}\})$
se il criterio di stop non si avvera, si procede scegliendo un test: $\text{Length} = long$
 - $t_1 \leftarrow \text{DT_Learner}(\{\text{Author}, \text{Thread}, \text{Where_read}\}, \text{User_action}, \{e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}\})$
 - tutti concordano sulla classe User_action
 - t_1 nodo-foglia *skips*
 - $t_0 \leftarrow \text{DT_Learner}(\{\text{Author}, \text{Thread}, \text{Where_read}\}, \text{User_action}, \{e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\})$
 - esempi non concordanti sulla classe
 - se lo stop non scatta si deve scegliere un nuovo test
 - ad es. $\text{Thread} = new$
 - continuando, alla fine t_0 (ramo $\text{Length} = short$) sarà:

```
if new(e) then reads
else
    if unknown(e) then skips else reads
```
- Risultato finale: albero / programma visto in precedenza

Dettagli da specificare:

- **criterio di stop**: condizioni esaurite o esempi con la stessa classe
 - *minimo* di esempi dei *nodi-figli*: stop sotto soglia minima
 - *minimo* di esempi *in un nodo*: stop se ci sono meno esempi della soglia
 - *miglioramento del criterio* di ottimizzazione: stop sotto soglia minima
 - ad es. IG
 - *profondità massima*: non si partiziona oltre tale profondità
- **output** restituito alle foglie
 - *stima puntuale*: dipende solo dalle condizioni nel percorso
 - *classificazione più probabile*:
mediana, media o distribuzione di probabilità sulle classi
- **scelta della condizione** per il partizionamento
 - quella che porta al più piccolo albero possibile
 - **scelta del test ottimale miope (*greedy*)**, come se fosse l'unico
 - massimizzando **IG** → ottimizzando verosimiglianza, entropia, log-loss..-

Esempio – Dataset precedente: scelta (miope) del test, massimizzando la likelihood ovvero minimizzando la log-loss

- prima del partizionamento, predizione ottimale basata sulla *frequenza empirica*:
 - 9 esempi di classe *reads* e 9 di classe *skips* → probabilità: 0.5
 - log-loss: $-(9 \cdot \log_2 0.5 + 9 \cdot \log_2 0.5)/18 = -(18 \cdot \log_2 0.5)/18 = 1$
- partizionando su *Author*:
[$e_1, e_4, e_5, e_6, e_9, e_{10}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}$] con *Author* = *known*
e [$e_2, e_3, e_7, e_8, e_{11}, e_{18}$] con *Author* = *unknown*
 - in ciascuna partizione stessa frequenza di esempi delle 2 classi
→ predizione ottimale: 0.5 per entrambe → log-loss: 1
 - nessuna diminuzione: il test non aiuta la predizione

- partizionando su *Thread*:

$[e_1, e_2, e_5, e_8, e_{10}, e_{12}, e_{14}, e_{15}, e_{17}, e_{18}]$ con $Thread = new$ e

$[e_3, e_4, e_6, e_7, e_9, e_{11}, e_{13}, e_{16}]$ con $Thread = followup$

- nella partizione *new*: 3 esempi per *skips* e 7 per *reads*
→ predizione *reads* (probabilità 7/10)
- nella partizione *followup*: 2 esempi per *reads* e 6 per *skips*
→ predizione *reads* (probabilità 2/8)
- log-loss *dopo* lo split:

$$-[3 \log_2(3/10) + 7 \log_2(7/10) + 2 \log_2(2/8) + 6 \log_2(6/8)]/18 \approx 15.3/18 \approx 0.85$$

- partizionando su *Length*:

$[e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}]$ con $Length = long$ e

$[e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}]$ con $Length = short$

- partizione *long*: tutti esempi con la stessa classificazione (*skips*)
→ predizione con probabilità 1
- partizione *short*: proporzione di 9 : 2 tra le classi
- log-loss: $-(7 \cdot \log_2 1 + 9 \cdot \log_2 9/11 + 2 \cdot \log_2 2/11)/18 \approx 7.5/18 \approx \underline{0.417}$
→ per la radice meglio *Length* (criterio miope)

Trattamento di *feature non booleane*:

1. Estensione a *partizionamento a più vie* (multiway split)

- per variabili multi-valore: un figlio per valore del dominio
- rappresentazione più complessa degli if-then-else
- **problemi:**
 1. valori della feature *senza* esempi di training
 2. euristiche miopi, e.g. IG, favoriscono variabili con *domini più numerosi*:
si conformano ai dati ma non garantiscono una rappresentazione compatta
 - ad es. una suddivisione a 4-vie equivale a 3 binarie (in profondità) con 4 figli

2. Partizionamento del dominio di una feature X in due *sottoinsiemi disgiunti* (come per quelle ordinali o per le variabili-indicatore)

- dominio di X totalmente ordinato → uso del *cut*
 - figli corrispondenti a $X \leq v$ e $X > v$ per un certo v
 - per scegliere v ottimale:
 - si ordinano gli esempi in base a X
 - si confrontano le possibili scelte di valori di taglio
- dominio di X discreto ma senza ordine naturale: sottoinsiemi arbitrari
 - se Y booleana, per ogni valore di X si determina la proporzione di esempi con $Y = \text{true}$
 - suddivisione miope *ottimale*, ordinando i valori in base a tale proporzione

3. Feature-target continue → alberi di regressione

- foglie: funzioni costanti o lineari basate sugli es. di training contenuti
- scelta del test in base ad altri criteri: e.g. *varianza*

Sovradattamento rispetto ai dati

(approfondito più avanti)

- si verifica quando si tenta di adattare il modello in base a criteri che valgono per i dati di training ma non in generale (dati futuri, di test)

Rimedi:

1. **fermare** il partizionamento, limitandolo ai casi in cui è davvero **utile**:
 - ad es. solo se l'errore sul training-set si riduca oltre una certa soglia
2. **potare** l'albero in caso di distinzioni tra esempi, **senza** garanzia di **generalità**

Esempio ↵ – Gioco: lancio di 2 monete; si vince se hanno la stessa faccia

Regressione e Classificazione Lineari

Algoritmi basati su modelli lineari per problemi di:

- regressione (caso continuo)
 - classificazione (caso discreto)
-

Regressione Lineare adattamento di una funzione *lineare* a training set con feature *numeriche*

- *feature*
 - input: X_1, \dots, X_n
 - target: Y
- *modello*: funzione lineare delle feature di input:

$$\hat{Y}_{\bar{w}}(e) = w_0 + w_1 \cdot X_1(e) + \cdots + w_n \cdot X_n(e) = \sum_{i=0}^n w_i \cdot X_i(e)$$

dove $\bar{w} = \langle w_0, w_1, \dots, w_n \rangle$ vettore di *pesi*

- forma compatta assumendo la feature aggiuntiva costante $X_0 = 1$

PROBLEMA DI OTTIMIZZAZIONE

Dato l'insieme di esempi E , errore quadratico su E per Y :

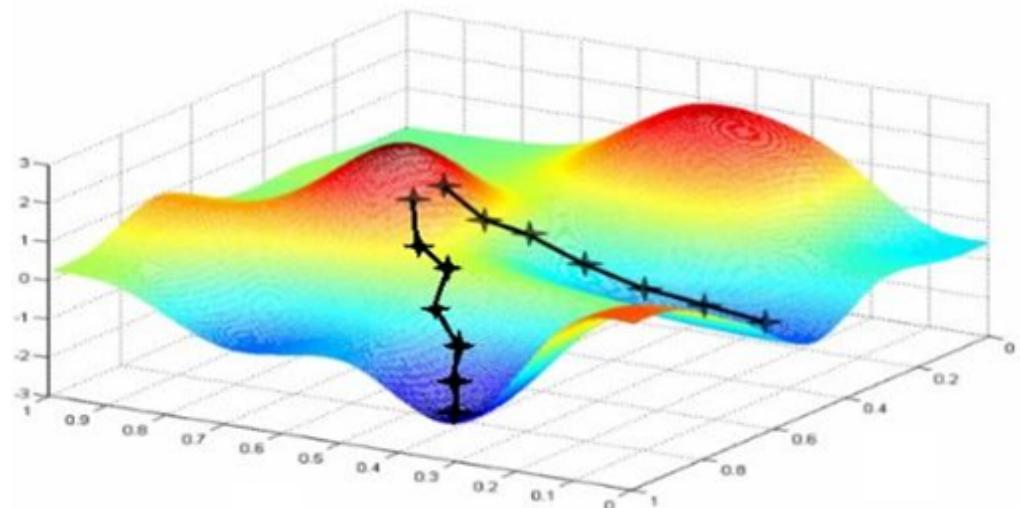
$$\begin{aligned} \text{error}(E, \bar{w}) &= \sum_{e \in E} (Y(e) - \hat{Y}_{\bar{w}}(e))^2 \\ &= \sum_{e \in E} \left(Y(e) - \sum_{i=0}^n w_i \cdot X_i(e) \right)^2 \end{aligned}$$

- caso lineare: \bar{w} ottimale calcolabile *analiticamente*
- calcolo *iterativo*: approccio più generale
 - usabile anche per l'errore di altre classi di funzioni (e misure da ottimizzare)

DISCESA DI GRADIENTE (DG)

Metodo **iterativo** per la ricerca (**locale**) dei minimi di funzioni d'errore o **loss**

- scelta (casuale) dei valori di \bar{w}
- si ripete:
 - decrementare ogni w_i in proporzione alla relativa derivata parziale:



$$w_i \leftarrow w_i - \eta \cdot \frac{\partial}{\partial w_i} \text{error}(E, \bar{w})$$

dove **η passo** o **learning rate**

(parametro dell'algoritmo)

- le derivate misurano l'influenza sull'errore di piccole variazioni dei pesi

MINIMIZZAZIONE DELL'ERRORE QUADRATICO

- per un modello lineare, errore quadratico funzione **convessa**
 - con **unico** minimo locale, quindi anche globale
 - con un passo sufficientemente piccolo, si converge al minimo locale
- derivate parziali di $\text{error}(E, \bar{w}) = \sum_{e \in E} (Y(e) - \hat{Y}_{\bar{w}}(e))^2$

$$\frac{\partial}{\partial w_i} \sum_{e \in E} \left(Y(e) - \sum_i w_i X_i(e) \right)^2 = \sum_{e \in E} -2 \cdot \delta(e) \cdot X_i(e)$$

- con $\delta(e) = Y(e) - \hat{Y}_{\bar{w}}(e)$
- aggiornamento dei pesi **dopo** aver iterato su **tutti** gli esempi
- in alternativa, aggiornamento **dopo ogni** esempio $e \in E$:

$$w_i \leftarrow w_i + \eta \cdot \delta(e) \cdot X_i(e)$$

- costante moltiplicativa **2** assorbita da η

procedure Linear_learner(X_s, Y, Es, η)

Input

X_s : insieme di feature input, $X_s = \{X_1, \dots, X_n\}$

Y : feature target

Es : insieme di esempi di training

η : learning rate

Output

funzione di predizione su esempi

Local

w_0, \dots, w_n : reali

inizializzare casualmente w_0, \dots, w_n

definire la funzione $pred(e) = \sum_i w_i \cdot X_i(e)$

repeat

for each $e \in Es$ **do**

$error \leftarrow Y(e) - pred(e)$

$update \leftarrow \eta \cdot error$

for each $i \in [0, n]$ **do**

$w_i \leftarrow w_i + update \cdot X_i(e)$

until terminazione

return $pred$

terminazione: 1) max n. passi; 2) errore ~0; 3) variazioni piccole di \bar{w}

VARIANTI DI DG

- **DG incrementale:** aggiornamento-pesi *dopo ogni esempio*:
 - si dovrebbero salvare tutti i cambiamenti e aggiornare dopo l'elaborazione di tutti gli esempi
- **DG stocastica:** aggiornamento incrementale con esempi selezionati casualmente
 - passi incrementali meno costosi: *convergenza* più veloce ma *non garantita*
 - singoli esempi possono allontanare i pesi dall'assegnazione che porta al minimo
- **DG a lotti:** aggiornamento-pesi dopo un *lotto (batch)* di esempi
 - modifiche calcolate dopo ogni esempio, ma applicate a *fine lotto*
 - lotto unico → DG di base
 - lotti con un solo esempio → DG incrementale
 - tipico partire con lotti piccoli per imparare più rapidamente e poi più grandi in vista della convergenza

DIVERSE FUNZIONI D'ERRORE

Diverse altre funzioni possibili,
anche *non ovunque differenziabili* o con derivata mai nulla:

- ***errore assoluto***: non differenziabile in 0
 - si può imporre che la derivata sia nulla:
l'errore ha un minimo e i pesi non devono cambiare
- ***altri tipi di errore***: situazione diversa
 - ad es. errore 0/1: derivata nulla (quasi ovunque) oppure non definita

CLASSIFICAZIONE: MODELLI CON FUNZIONI LINEARI APPIATTITE

Classificazione binaria: $\{0, 1\}$ come dominio di Y

- in caso di più feature-obiettivo: apprendimento *separato*

Funzioni *lineari* poco utili:

- non servono predizioni al di fuori di $[0, 1]$

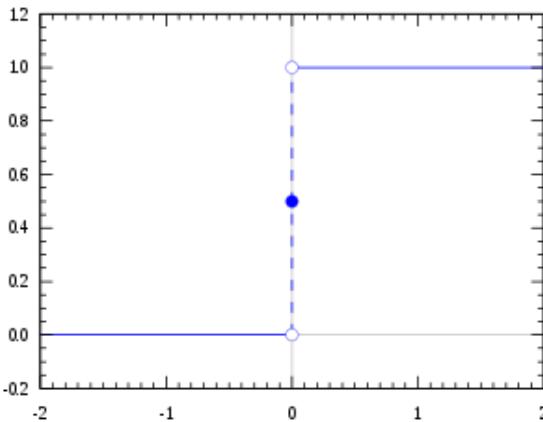
Per ottenere un classificatore, funzione **lineare appiattita (squashed)**:

$$\begin{aligned}\hat{Y}_{\bar{w}}(e) &= f(w_0 + w_1 \cdot X_1(e) + \cdots + w_n \cdot X_n(e)) \\ &= f\left(\sum_i w_i \cdot X_i(e)\right)\end{aligned}$$

- $f : \mathbb{R} \rightarrow R$, **funzione di attivazione**
 - con $R \subseteq \mathbb{R}$, ad es. $R = [0, 1]$

Step o scalino⁶ funzione di attivazione semplice:

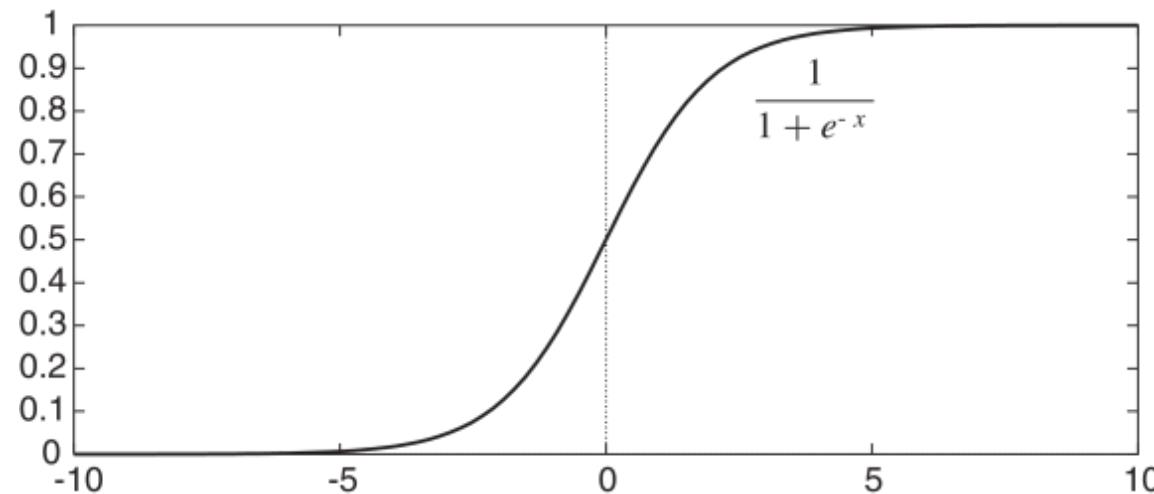
$$step_0(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$



- alla base del **percettrone** [5]
- difficile adattare DG a funzioni non ovunque differenziabili
 - con una funzione quasi ovunque differenziabile come *step*, l'ampiezza dello scalino dovrebbe tendere a zero per garantire la convergenza

Sigmoide o funzione logistica:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



- appiattisce le predizioni portandole nell'intervallo $]0, 1[$ appropriato ai fini della classificazione
- differenziabile, con derivata:

$$\frac{d}{dx} \text{sigmoid}(x) = \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x))$$

REGRESSIONE LOGISTICA

Modello di **classificazione**: funzione lineare appiattita dal sigmoide

$$\hat{Y}(e) = \text{sigmoid} \left(\sum_{i=0}^n w_i \cdot X_i(e) \right)$$

- per determinare i **pesi \bar{w}** si minimizza la **log-loss**, ossia la **log-likelihood negativa**:

$$LL(E, \bar{w}) = - \left[\sum_{e \in E} Y(e) \log \hat{Y}(e) + (1 - Y(e)) \log(1 - \hat{Y}(e)) \right]$$

- con derivate parziali:

$$\frac{\partial}{\partial w_i} LL(E, \bar{w}) = \sum_{e \in E} -\delta(e) \cdot X_i(e)$$

- dove $\delta(e) = Y(e) - \hat{Y}_{\bar{w}}(e)$ e costante moltiplicativa 2 assorbita in η

Algoritmo analogo al precedente, adattando la funzione di predizione:

procedure Logistic_regression_learner(X_s, Y, E_s, η)

Input

X_s : insieme di feature di input, $X_s = \{X_1, \dots, X_n\}$

Y : feature obiettivo

E_s : insieme degli esempi di training

η : learning rate

Output

funzione di predizione per gli esempi

Local

w_0, \dots, w_n : reali

Inizializzare w_0, \dots, w_n casualmente

definire $pred(e) = sigmoid(\sum_i w_i \cdot X_i(e))$

repeat

for each $e \in E_s$ in ordine casuale **do**

$error \leftarrow Y(e) - pred(e)$

$update \leftarrow \eta \cdot error$

for each $i \in [0, n]$ **do**

$w_i \leftarrow w_i + update \cdot X_i(e)$

until terminazione

return $pred$

Esempio – Dato il problema dei post visto in precedenza, determinare un classificatore basato su funzione lineare appiattita

- classificatore corretto degli esempi:

$$\widehat{\text{Reads}}(e) = \text{sigmoid}(-8 + 7 \cdot \text{Short}(e) + 3 \cdot \text{New}(e) + 3 \cdot \text{Known}(e))$$

trovato dopo circa 3000 iterate dell'algoritmo con $\eta = 0.05$

- $\widehat{\text{Reads}}(e)$ vero (valore predetto per e più vicino a 1 che a 0) se $\text{Short}(e)$ vero e o $\text{New}(e)$ o $\text{Known}(e)$ veri
- analoga alla funzione appresa attraverso gli alberi di decisione

Minimizzazione dell'**errore quadratico** (invece della LL)

→ diversa derivata parziale:

- aggiornamento nell'algo. precedente:

$$\text{update} \leftarrow \eta \cdot \text{error} \cdot \text{pred}(e) \cdot (1 - \text{pred}(e))$$

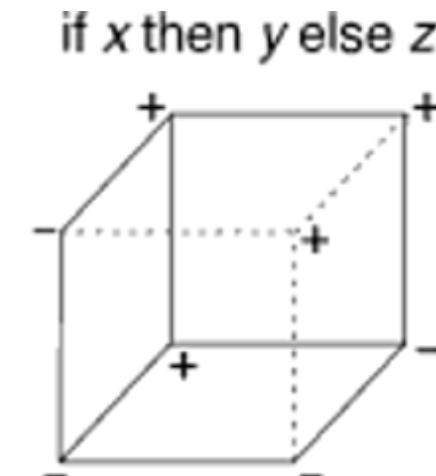
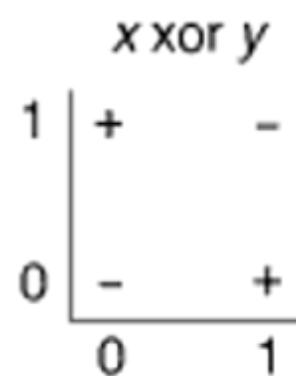
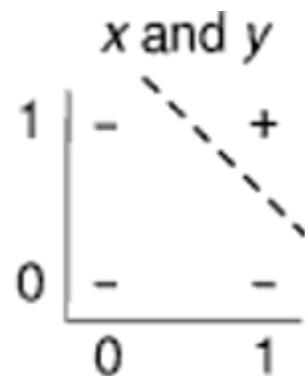
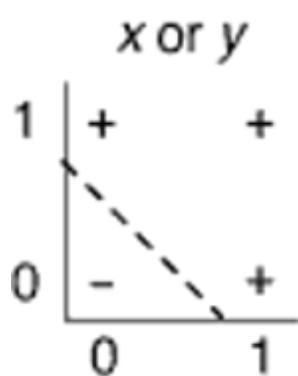
SEPARABILITÀ LINEARE

In uno spazio n -dimensionale, i.e. con n feature (di input):

- **iperpiano:** insieme di punti che soddisfano un vincolo espresso come equazione **lineare** sulle variabili
 - determina uno spazio con $(n - 1)$ dimensioni
 - caso 2D: retta; caso 3D: un piano, ecc.
- **problema di classificazione linearmente separabile:** esiste un iperpiano che separi gli esempi delle due classi
 - con `Logistic_regression_learner` su tali problemi:
 - errore riducibile arbitrariamente
 - iperpiano: insieme di punti $\sum_i w_i \cdot X_i = 0$
 - \bar{w} pesi appresi
 - da un lato, predizione maggiore di 0.5, dall'altro minore di 0.5

Esempio – Separabilità lineare

- problemi linearmente separabili: or e and
- non linearmente separabili
 - xor (*or esclusivo*)
 - nessuna retta può separare le due classi di esempi
 - non rappresentabile con un classificatore lineare
 - if x then y else z funzione su feature di input booleane x , y e z



Separabilità lineare spesso non facilmente determinabile a priori

Esempio – Classificazione con modello *lineare* (valore *lin*)

- località-vacanze descritte da: presenza di eventi culturali, disponibilità voli, clima caldo, presenza di musica e attrazioni naturali
- *predizione* di *Likes* con rappresentazione numerica: [0, 1]

<i>Culture</i>	<i>Fly</i>	<i>Hot</i>	<i>Music</i>	<i>Nature</i>	<i>Likes</i>	<i>lin</i>	\widehat{Likes}
0	0	1	0	0	0	-9.09	0.00011
0	1	1	0	0	0	-9.08	0.00011
1	1	1	1	1	0	-4.48	0.01121
0	1	1	1	1	0	-6.78	0.00113
0	1	1	0	1	0	-2.28	0.09279
1	0	0	1	1	1	4.61	0.99015
0	0	0	0	0	0	0.01	0.50250
0	0	0	1	1	1	2.31	0.90970
1	1	1	0	0	0	-6.78	0.00113
1	1	0	1	1	1	4.62	0.99024
1	1	0	0	0	1	2.32	0.91052
1	0	1	0	1	1	0.01	0.50250
0	0	0	1	0	0	-4.49	0.01110
1	0	1	1	0	0	-11.29	0.00001
1	1	1	1	0	0	-11.28	0.00001
1	0	0	1	0	0	-2.19	0.10065
1	1	1	0	1	0	0.02	0.50500
0	0	0	0	1	1	6.81	0.99890
0	1	0	0	0	1	0.02	0.50500

(cont.)

- con $\eta = 0.05$, dopo 10000 iterate, modello di predizione (approssimata):

$$\begin{aligned} lin(e) &= 2.3 \cdot Culture(e) + 0.01 \cdot Fly(e) - 9.1 \cdot Hot(e) + \\ &\quad - 4.5 \cdot Music(e) + 6.8 \cdot Nature(e) + 0.01 \end{aligned}$$

$$\widehat{Likes}(e) = \text{sigmoid}(lin(e))$$

- ok per tutti gli esempi tranne quattro
- valore predetto vicino a 0.5
- modello **stabile** con altre inizializzazioni dei pesi
 - con più iterate, aumenta l'accuratezza (si abbassa l'errore)
 - ma non per i 4 casi di cui sopra ← dataset **non linearmente separabile**

PROBLEMI MULTI-CLASSE

- Variabile target → **variabili-indicatore** binarie
 - modelli da apprendere separatamente
- Per ottenere la predizione del target originario: **combinazione** dei classificatori
 - vincolo: per ogni esempio, una sola Y_i true
 - il modello non dovrebbe predire *true* per più di una, o nessuna
- **Predizione finale**
 - q_1, \dots, q_k valori predetti per Y_1, \dots, Y_k , con $q_i \geq 0$
 - un modello della **distribuzione delle probabilità** può predire $Y = y_i$ con probabilità $\frac{q_i}{\sum_j q_j}$
 - si può quindi predire la **moda**: y_i col massimo q_i

SOVRADATTAMENTO

Sovradattamento (*Overfitting*): apprendimento di modelli basati su *regolarità apparenti* presenti solo nel training set, ma assenti altrove (nel dominio reale)

- tipico nella ricerca di *segnali* in una distribuzione casuale

1. correlazioni *spurie* nei dati:

- non si riflettono nell'intero dominio del problema

2. *eccesso di fiducia* nelle predizioni del modello:

- rispetto a quanto può essere autorizzato dai dati a disposizione
- emerge lavorando sui dati di test

Esempio – Sito con giudizi sui ristoranti: $1 \rightarrow 5$ stelle

- si vogliono evidenziare ristoranti *migliori*
 - ristoranti con *molti* giudizi:
difficile tenere una media di 5 stelle,
richiede l'unanimità
 - ristoranti con *un solo* giudizio e pari a 5 stelle:
apparentemente tra i migliori ma improbabile che lo siano davvero
- analogo problema per i ristoranti con un solo rating minimo

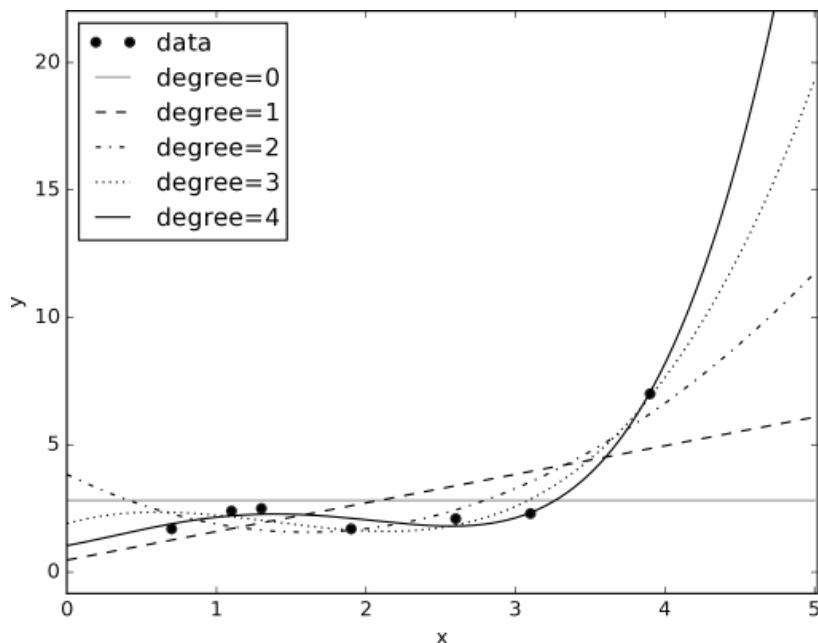
SOVRADATTAMENTO: CAUSE

- Fenomeno legato alla cosiddetta *regressione verso la media*⁷
→ valori determinati da qualità e caso
 - più dati abbassano la casualità
- Dipende anche dalla *complessità del modello*:
 - modello più complesso, con più parametri,
più facile da adattare ai dati di uno più semplice
 - *caso estremo*: un parametro per ogni esempio

Esempio – Polinomio di grado k :

$$y = w_0 + w_1 \cdot x + w_2 \cdot x^2 + \cdots + w_k \cdot x^k$$

- con un algoritmo per modelli **lineari** come DG:
 - determina \vec{w} (pesi) che minimizzano l'**errore quadratico**
 - **feature target**: valore di y
 - **feature di input**: $1, x, x^2, \dots, x^k$
 - polinomi fino a $k = 4$ per dati visti in precedenza:



(..cont.)

- polinomi di grado **maggiore** più adattabili ai dati ma non necessariamente migliori: più **estremi** nell'estrapolazione
 - tendono asintoticamente verso $\pm\infty$ (tranne quello costante)
 - massimo k per cui $w_k \neq 0$ pari/dispari
→ per $x \rightarrow \pm\infty$ predizioni con segno \pm/\mp
 - il polinomio di grado 4 tende a $-\infty$ al diminuire di x : ragionevole?
- per polinomi appresi tramite DG, scelta del **passo** importante:
 - x vicino a 0 ($|x| \ll 1$) → x^k è piccolissimo
 - x grande ($|x| \gg 1$) → x^k enorme
 - x espresso in **mm** (e.g. $x(e_7) = 25$)
→ coefficiente di x^4 con grande impatto sull'errore
 - x espresso in **m** ($x(e_7) = 0.025$) impatto molto piccolo

Osservazioni

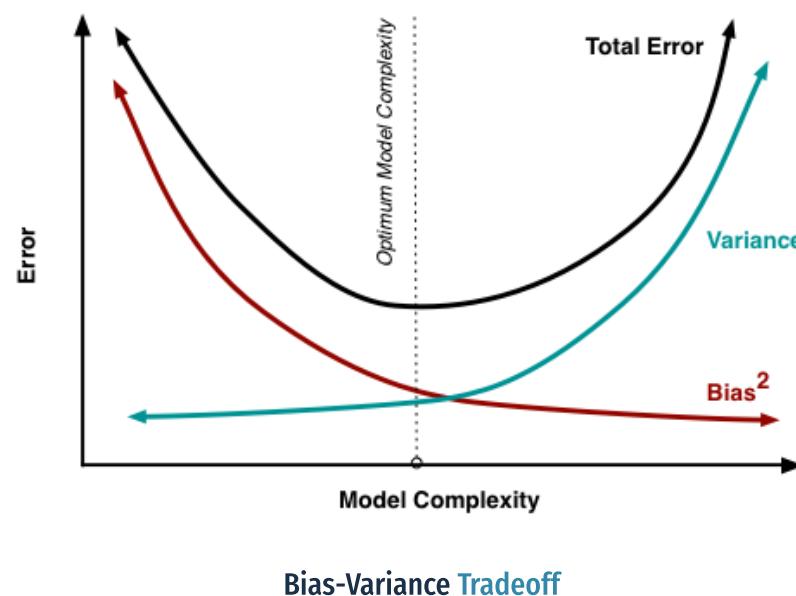
- più complesso il modello → più facile che si verifichi il sovradattamento
- più dati → migliori predizioni
 - *big data*: crescono il numero di feature e il numero di esempi
 - ad es. una descrizione dettagliata di un paziente (molte feature) può essere ritenuta unica al mondo

Errore sul test set

cfr. [7]

$$Errore_2(\text{Test}) = \text{Bias}^2 + \text{Varianza} + \text{Errore_irriducibile}$$

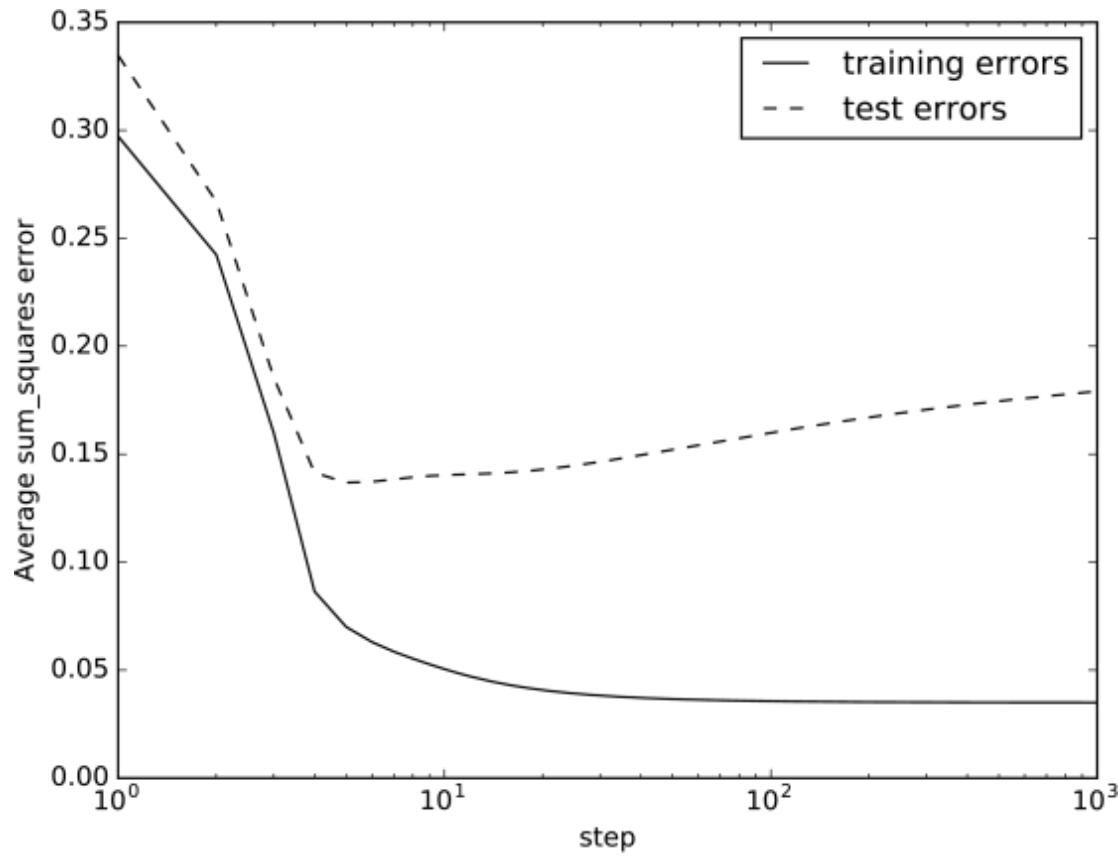
causato da:



- **bias**: errore dovuto all'*imperfezione* del modello appreso
 - basso quando il modello è vicino all'obiettivo ideale:
processo reale che governa la generazione dei dati (**ground truth**)
 - decomponibile in:
 - **bias di rappresentazione**: la rappresentazione non contiene un'ipotesi vicina alla *ground truth*
 - **bias di ricerca**: l'algoritmo non cerca esaustivamente lo spazio di ipotesi
 - **Esempi**
 - alberi di decisione: bias di rappresentazione basso (si può rappresentare qualunque funzione) ma bias di ricerca crescente con il numero di feature (moltissimi alberi)
 - regressione lineare: se risolta analiticamente, grande bias di rappresentazione ma bias di ricerca nullo (maggiore usando la **DG**)

- **varianza:** errore dovuto alla limitatezza dei dati disponibili
 - modello più complesso (con più parametri) richiede più dati
 - se il quantitativo di dati è fisso: compromesso **bias-varianza**⁸
 - modello complesso e preciso ma necessario quantitativo di dati non disponibile
→ bias basso e alta varianza
 - modello semplice [non accurato], parametri facilmente stimabili con i dati disponibili
→ bassa varianza ma bias alto
- **rumore:** errore *intrinseco* dovuto a:
 - dati che dipendono da *feature non considerate* nel modello
 - *naturale aleatorietà* nel processo di generazione dei dati

Esempio – Errori (quadratico) nelle numero di iterazioni nella DG



- errore sul training set, decresce al crescere del numero di iterate
- errore sul test set, dopo aver raggiunto un minimo torna a crescere
 - adattando il modello agli es. di training si tende ad avere più fiducia in un modello imperfetto, per cui l'errore sul test set aumenta

SOVRADATTAMENTO: RIMEDI

Pseudoconteggi medie e mancanza di dati (regressione verso la media)

Regolarizzazione compromesso esplicito tra complessità del modello e adattamento ai dati

Cross Validation uso di parte dei dati per testare il modello appreso, individuando l'eventuale sovradattamento

Pseudoconteggi

Media predizione ottimale rispetto a molte misure di predizione

- **media empirica** su E non sempre valida per stime riguardanti nuove istanze
 - ad es. un valore non osservato in E avrebbe probabilità nulla
 - considerando Y_v booleana, probabilità dalla media/freq. dei casi di $Y_v = 1$

Esempio

- **predizione del voto nel prossimo esame:**
 - **media appropriata** se diversi esami già sostenuti
 - **media non appropriata** se basata su un solo esame (**indefinita con nessuno**)

MEDIA MOBILE

Disponendo dei valori osservati v_1, \dots, v_n , si voglia predire il successivo: \hat{v}

- media (*average*) dei primi n valori:

$$\begin{aligned} a_n &= \frac{v_1 + \cdots + v_{n-1} + v_n}{n} = \frac{v_1 + \cdots + v_{n-1}}{n} + \frac{v_n}{n} \\ &= \frac{v_1 + \cdots + v_{n-1}}{n} \cdot \frac{n-1}{n-1} + \frac{v_n}{n} = a_{n-1} \cdot \frac{n-1}{n} + \frac{v_n}{n} \\ &= a_{n-1} + \frac{v_n - a_{n-1}}{n} \end{aligned}$$

media mobile (*running average*): media corrente dei dati osservati

- *aggiornamento* per un nuovo valore v :

dati n valori osservati con media corrente a ,

$$n \leftarrow n + 1 \text{ e } a \leftarrow a + (v - a)/n$$

medie *iniziali*?

Rimedio al problema della *media/probabilità iniziale nulla*:

- **pseudoconteggi** da conoscenza pregressa (*prior knowledge*)
 - conteggio su valori iniziali ai quali aggiungere i dati di training
→ **smoothing additivo**⁹

Per $n = 0$ si assume una media iniziale a_0 (non ricavabile dai dati)

- predizione:

$$\hat{v} = \frac{v_1 + \cdots + v_n + c \cdot a_0}{n + c}$$

con $c \in \mathbb{R}$ **pseudoconteggio** del numero di dati iniziali non osservati

- c controlla la quantità di *regressione verso la media*
 - $c = 0 \rightarrow$ predizione \equiv media empirica
- implementato nella **media mobile** inizializzando a con a_0 e n con c

Esempio – Caso dei ristoranti: stima della media dei giudizi *futuri* (test)

- Si assume che nuovi dati somiglieranno ai dati già acquisiti
 - → nuovo ristorante simile a uno medio (non sempre una buona assunzione)
- Prima di raccogliere rating, ragionevole adottare un giudizio medio iniziale a_0
- Per stimare c , volendo fare predizioni accurate su ristoranti di *fascia top*
 - si considera un ristorante in tale fascia con 5★ come unico giudizio ricevuto
 - dovrà essere come gli altri della fascia
 - sia a' il rating medio dei ristoranti della fascia
 - pesato sul numero di rating 5★ di ciascun ristorante
 - perché tale ristorante confermi la media a' della fascia:
$$a' = a_0 + (5 - a_0)/(c + 1)$$
 - se $a_0 = 3$ e $a' = 4.5$, allora: $4.5 = 3 + (5 - 3)/(c + 1)$, da cui $c = \frac{1}{3}$
 - se fosse $a' = 3.5 \rightarrow c = 3$

Esempio ← – Si seleziono un numero a caso $p \in [0, 1]$,
ground truth per $P(Y = 1)$ (Y booleana)

- si generino n esempi di training con $P(Y = 1) = p$ per
 $n = 1, 2, 3, 4, 5, 10, 20, 100, 1000$
- con n_1 campioni con $Y = 1 \rightarrow n_0 = n - n_1$ con $Y = 0$
- **problema:** imparare da n_0 e n_1 uno stimatore \hat{p} per predire nuovi casi
 - casi di test (ad es. 100) generati a partire da p
 - un buon \hat{p} dovrà minimizzare l'errore sui casi di test
 - 1000 ripetizioni per farsi un'idea
- $\hat{p} = n_1/(n_0 + n_1)$ massimizza la log-likelihood ma funziona male:
 - caso $n_0 = 0$ (o $n_1 = 0$), ma 0 (o 1) appare nel test set → likelihood nulla
- con arrotondamento di Laplace: $\hat{p} = (n_1 + 1)/(n_0 + n_1 + 2)$
→ massima likelihood di tutti gli estimatori sul test set
 - meglio del precedente anche in termini di errore quadratico
- **NB** distribuzione di p non uniforme → non il miglior predittore

Regolarizzazione

*Rasoio di Occam*¹⁰ preferire modelli semplici a quelli complessi

Regolarizzatore: termine da aggiungere alla misura di adattamento ai dati in modo da *penalizzare* la complessità dell'ipotesi da apprendere

Forma: scelta dell'ipotesi h che minimizzi

$$\sum_e \text{error}(e, h) + \lambda \cdot \text{regularizer}(h)$$

- $\text{error}(e, h)$ misura la qualità dell'adattamento su e
- $\text{regularizer}(h)$: penalizza complessità del modello e deviazione media
 - **parametro di regolarizzazione** λ , media tra adattamento e semplicità (o deviazione della media)
 - serve anche a sommare quantità su scale diverse
 - determinabile in base a BK o tramite *cross-validation*
 - **molti** esempi: la somma a sinistra tende a dominare sul regolarizzatore
 - **pochi** esempi: il regolarizzatore ha maggiore effetto

MODELLI E REGOLARIZZATORI

Alberi di decisione:

- misura di **complessità**: numero di test
 - dipende dal numero delle foglie per un albero binario
- nella minimizzazione dell'errore quadratico
si aggiunge una funzione delle dimensioni dell'albero:

$$\left(\sum_{e \in E} (Y(e) - \hat{Y}(e))^2 \right) + \lambda \cdot |albero|$$

dove $|albero|$ è pari alla misura sopra indicata

- un test in un nodo è **utile** se riduce l'errore di λ

Modelli con parametri a valori reali:

- un regolarizzatore L_2 penalizza la somma dei quadrati dei parametri
- nella **ridge regression** viene sommato all'**errore quadratico**:

$$\left[\sum_{e \in E} \left(Y(e) - \sum_{i=0}^n w_i \cdot X_i(e) \right)^2 \right] + \lambda \sum_{i=0}^n w_i^2$$

- con un regolarizzatore $L_1 \rightarrow$ LASSO [7]

Regressione logistica:

- Per minimizzare la **log-loss** con un regolarizzatore L_2 :

$$-\left[\sum_{e \in E} (Y(e) \log \hat{Y}(e) + (1 - Y(e)) \log(1 - \hat{Y}(e))) \right] + \lambda \left(\sum_{i=0}^n w_i^2 \right)$$

dove $\hat{Y}(e) = \text{sigmoid}(\sum_{i=0}^n w_i \cdot X_i(e))$

Implementazione della *Regolarizzazione L_2* in Linear_learner e Logistic_regression_learner

- si introduce il passo:

$$w_i \leftarrow w_i - \eta \cdot (\lambda/|E|) \cdot w_i$$

- diviso per $|E|$ perché operato *su ciascun esempio*
- è anche possibile regolarizzare *dopo ogni iterata* su tutti gli esempi
 - in tal caso non va diviso
 - $\eta\lambda/|E|$ costante quindi pre-calcolabile

Regolarizzatore L_1 : somma dei valori assoluti dei parametri

- con la log-loss si minimizza:

$$-\left(\sum_{e \in E} (Y(e) \log \hat{Y}(e) + (1 - Y(e)) \log(1 - \hat{Y}(e)))\right) + \lambda \left(\sum_{i=0}^n |w_i|\right)$$

- derivata parziale del termine: $sign(w_i) = w_i / |w_i| \in \{-1, +1\}$
 - non definito in 0, ma non serve spostarsi da 0: già valore minimo
- implementato facendo tendere a 0 ogni parametro di una costante
 - diventa nullo nel caso si cambi il segno del parametro
- iterative soft-thresholding: nella DG per la regressione logistica:

$$w_i \leftarrow sign(w_i) \cdot \max(0, |w_i| - \eta \cdot \lambda / |E|)$$

- in caso di feature numerose, il regolarizzatore L_1 tende ad annullare molti pesi, facendo ignorare le relative feature → selezione delle feature

Problema difficile determinare la complessità giusta dei modelli da impiegare **prima** di aver visto i dati

- dipende dal numero di parametri e/o da altri **iperparametri** dell'algoritmo

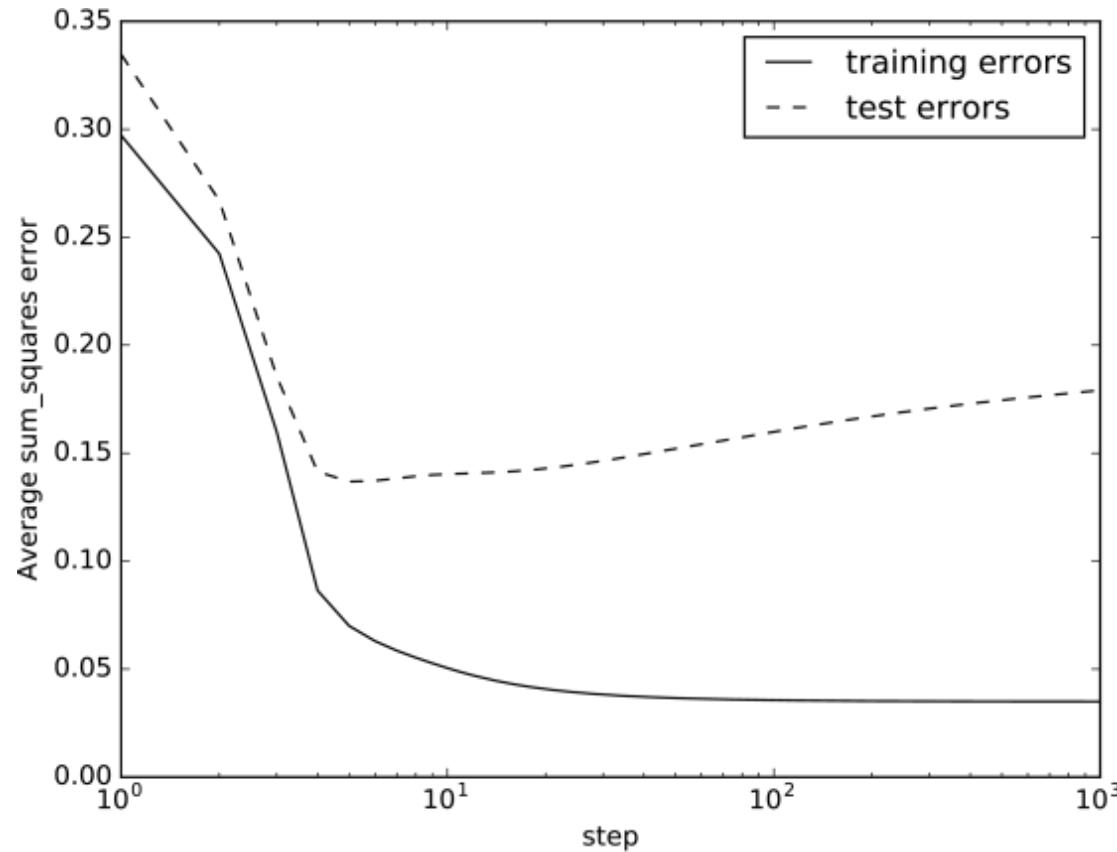
Cross Validation si trattiene *parte* dei dati di training per valutare un modello appreso sulla base del resto degli esempi

- si dividono i dati in due parti:
 1. insieme di **training**
 - per l'apprendimento dei modelli
 2. insieme di **validazione** (**validation set**)
 - per determinare il miglior modello
- CV da ripetere scambiando i ruoli degli esempi per ottenere modelli diversi
- alla fine si sceglierà il modello migliore



Si usa anche ottenere diversi partizionamenti in training/test-set su cui apprendere/valutare modelli

Esempio - Alberi, andamento dell'*errore* quadratico:



- **training**: diminuisce al crescere dell'albero
- **test**: diminuisce fino ad un certo punto, poi comincia a risalire

Obiettivo CV mira a una scelta dei parametri / rappresentazione di una h per la quale l'errore sia **minimo** sul set di validazione

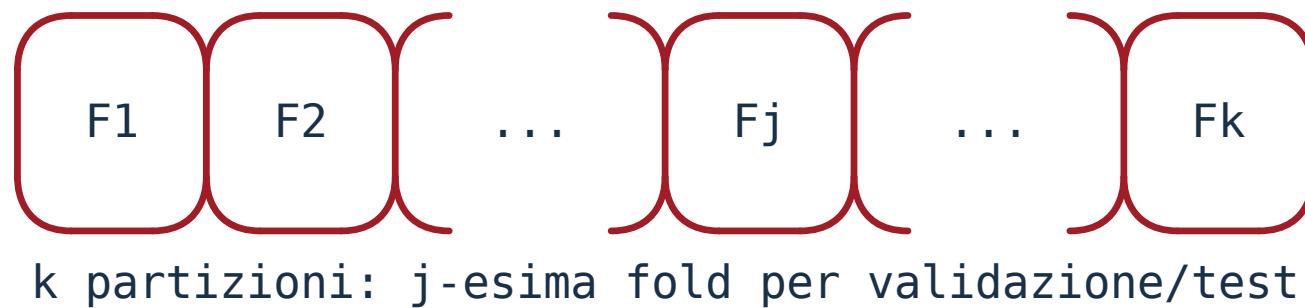
- ipotizzando che questo possa valere anche per il test set
- validation set \neq test set
 - test set non va usato nel learning
 - serve alla predizione su esempi non considerati in precedenza
 - validation set: surrogato dei futuri esempi



tipicamente l'addestramento ha bisogno di molti esempi per avere modelli migliori, ma ciò non deve andare a scapito del set per la validazione

K-FOLD CROSS VALIDATION

Riuso degli esempi per il training e la validazione
per regolare i parametri (quindi la complessità) del modello appreso

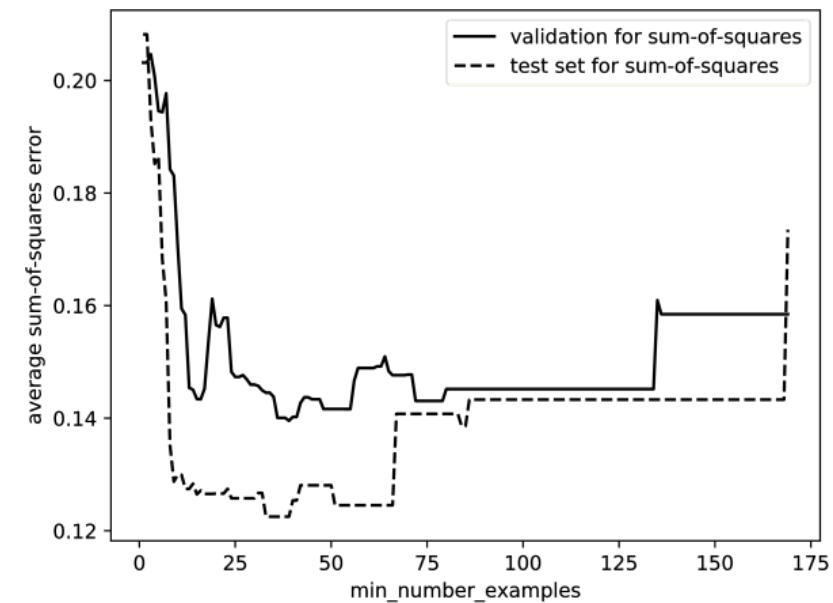


Passi (per valutare una scelta di parametri)

- suddividere casualmente il dataset in k parti di uguali dimensioni, *fold*
- valutare le impostazioni: addestrando il modello k volte,
ognuna con una distinta fold per la validazione
 - e.g. ripetendo per $k = 10$ volte: training su 90% e validazione sul 10% dei dati
 - la validazione coinvolge ogni esempio in *una* fold
- determinare e restituire le impostazioni ottimali

Esempio – parametro degli alberi di decisione:

- numero minimo di esempi *min_number_examples* per un nuovo nodo interno, da cui dipende il *criterio di stop*
 - soglia troppo bassa: tendenza al *sovradattamento*
 - soglia troppo alta: tendenza a *non generalizzare*
- errore sul set di validazione per una 5-fold CV in funzione di *min_number_examples*:
 - per ogni punto delle ascisse, l'algoritmo è stato fatto girare 5 volte per poi valutare l'errore medio quadratico sul set di validazione
 - minimo a 39: miglior valore per tale parametro
 - viene mostrato anche l'errore sul test set per alberi ottenuti con varie impostazioni del parametro:
 - 39 ragionevole anche in base al test set



LEAVE-ONE-OUT CROSS VALIDATION

k -fold con k pari al numero di esempi di training

- caso estremo per quando si hanno pochi dati
- con n esempi di training, si apprendono n modelli
 - per ogni esempio e :
 - usa tutti gli altri per addestrare un modello
 - valuta il modello su e
- la complessità cresce col numero degli esempi di training
 - poco pratico con modelli prodotti in fasi di training *indipendenti*
 - OK se il modello creato in una esecuzione può essere *modificato* in modo efficiente nel successivo, sostituendo un solo esempio con un altro

RETI NEURALI E APPRENDIMENTO PROFONDO

Reti Neurali Artificiali

Reti Neurali Artificiali (RNA/ANN) ispirate dai neuroni cerebrali:

- **unità artificiali diverse dai corrispettivi biologici**
 - molto più semplici e meno numerose dei 10^{11} del cervello umano
- **popolari per problemi che comportano un ragionamento di *basso livello*, con molti dati disponibili**
 - *image interpretation, speech recognition e machine translation*
 - molto **flessibili** e capaci di **inventare** nuove feature

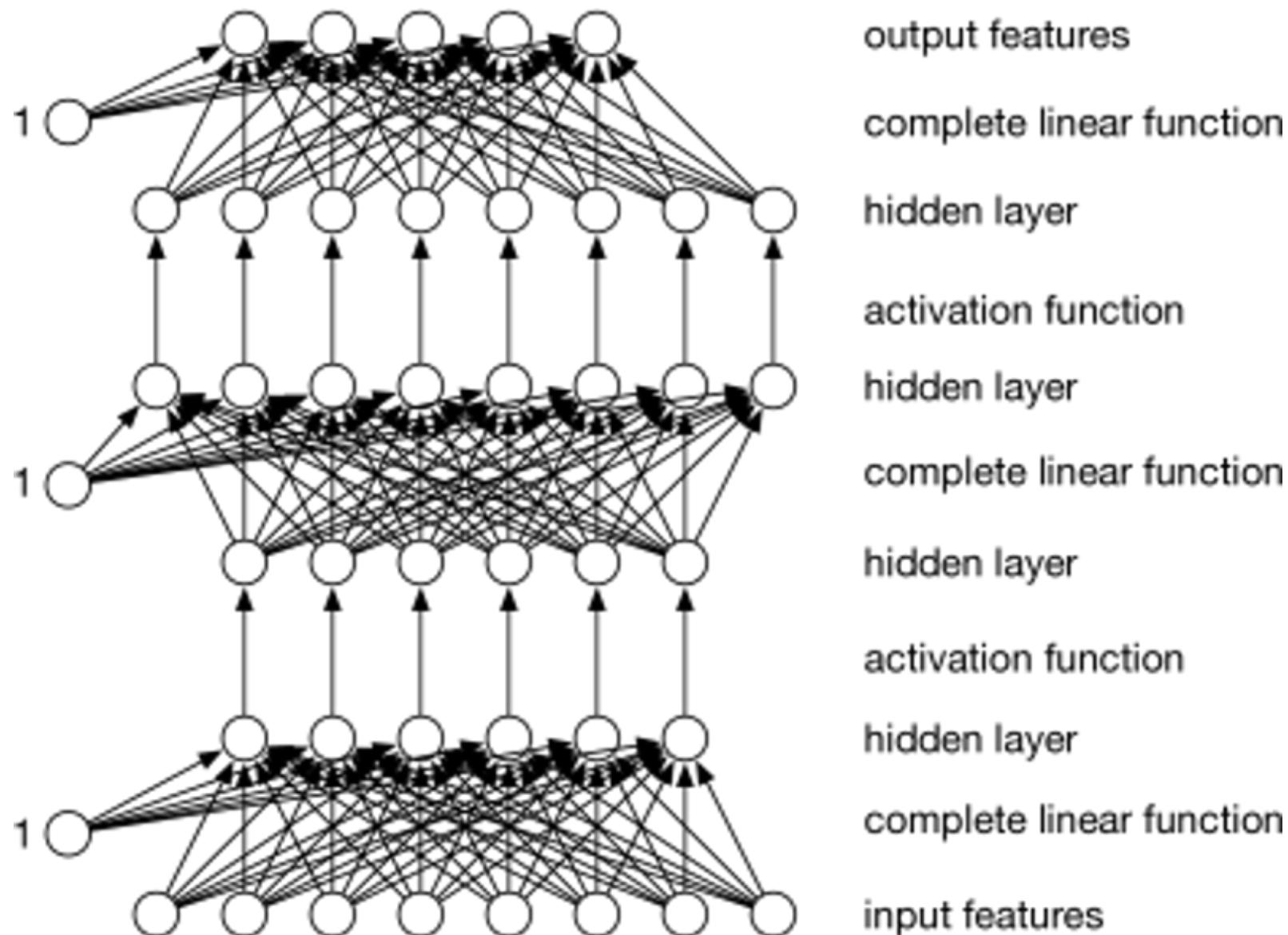
Motivi d'interesse:

- nelle neuroscienze, per comprendere i sistemi naturali
 - simulazione di sistemi propri di animali semplici (comportamento)
- comprensione dei **meccanismi** funzionali e astratti **dell'intelligenza**
 - **ipotesi**: replicare i meccanismi del cervello per realizzarne le funzionalità
 - testata confrontando forme d'intelligenza con e senza tali meccanismi
 - contro-ipotesi: gli aerei usano gli stessi principi del volo degli uccelli ma non lo stesso meccanismo
- nuovi **modelli computazionali**
 - convenzionali: pochi processori molta memoria (per lo più inerte)
 - nel cervello: moltissimi processi asincroni distribuiti, senza un master
 - RN spesso implementate su architetture massivamente **parallele**
- diversa misura di complessità / **learning bias** rispetto ad altri modelli:
 - RN possono rappresentare qualsiasi funzione su feature discrete
 - come gli alberi di decisione
 - funzioni rappresentate da RN non sempre realizzabili attraverso alberi

Reti Neurali Gerarchia di funzioni lineari **feed-forward** inframmezzate a funzioni di attivazione

In genere RN con diverse feature di input ma anche più feature-target:

- feature a valori *reali*
 - quelle discrete trasformabili in variabili-indicatrici o feature ordinali
- le unità di *input* alimentano *strati nascosti* di unità / feature
 - non osservate direttamente ma utili alla predizione
 - ogni unità prevede una funzione semplice che combina output di unità di strati inferiori
- alla fine gli strati nascosti vanno ad alimentare le predizioni degli *output*



ARCHITETTURA

- Ogni esempio ha un valore per ogni unità
- Ogni layer/strato di unità opera in funzione dei precedenti → **tipi**:
 - Layer di **input**: un'unità per feature di input
 - strato avvalorato da un esempio
 - Layer **completo lineare**: $o_j = \sum_i w_{ji} v_i$,
output o_j in funzione (lineare) dei valori in input v_i allo strato
 - peso w_{ji} da apprendere, per ogni (arco) coppia input-output del layer
 - in aggiunta, nodo con la costante 1 menzionata
 - **NB** layer lineari **consecutivi** trasformabili in un unico layer lineare
 - Layer di **attivazione**: $o_i = f(v_i)$,
output o_i come funzione del corrispondente valore in input, v_i
 - f funzione di attivazione, tipicamente:
 - **Sigmoidale**: $f(x) = 1/(1 + e^{-x})$
 - **Rectified Linear Unit (ReLU)**: $f(x) = \max(0, x)$
 - f dovrebbe essere (quasi ovunque) differenziabile

PREDIZIONE

- **Regressione:** ultimo layer lineare completo
 - per avere l'intero dominio dei valori
- **Classificazione binaria:** output mappati su $\{0, 1\}$
 - tipicamente tramite una funzione sigmoidale dei suoi input

BACK-PROPAGATION

Passo di DG stocastica su tutti i pesi:

- per ogni esempio e , si aggiorna ogni peso w in ragione di $\frac{\partial}{\partial w} \text{error}(e)$
 - usando le proprietà¹¹ delle derivate:
 - regole di decomposizione lineare e di concatenazione (per le composte)

Apprendimento in due passate (per ogni esempio)

- **Predizione**: per ciascun layer, dati i valori in input,
si calcola un valore per l'output
- **BACK-PROPAGATION (BP)**: andando a ritroso attraverso i layer,
si aggiornano tutti i pesi (nei layer lineari) della rete

MODULARIZZAZIONE

Layer: modulo indipendente che implementa

- le due passate
- l'aggiornamento dei pesi
- l'invio di un termine relativo all'errore ai livelli inferiori:
 - BP calcola la derivata per tutti i pesi – regola di composizione – in un'unica passata a ritroso sulla rete

(Virtualmente) alla rete può essere aggiunto un ***layer d'errore***

- dopo quello di output

- Per ogni esempio errore al layer finale in base alle differenze:

- per la j -ma feature-target:
 - $values[j]$ **output** predetto dal layer finale
 - $Y[j]$ valore disponibile dal training-set
- **errore quadratico** per un dato esempio:

$$error = \sum_j (values[j] - Y[j])^2$$

- per un generico peso w : $values[j]$ dipende da w mentre $Y[j]$ costante:

$$\frac{\partial}{\partial w} error = \sum_j 2(values[j] - Y[j]) \frac{\partial}{\partial w} values[j]$$
- errore **retro-propagato**: $Y[j] - values[j]$ 2 assorbito nel passo η
- BP aggiorna ogni w quando è stato calcolato questo valore per ogni w
- **NB** negazione della derivata \rightarrow **ascesa di gradiente**:
 - errore **positivo**: valore da incrementare
 - errore **negativo**: valore da decrementare
- altre funzioni d'errore, e.g. log-loss \rightarrow errori iniziali diversi

- Per ogni layer, termine d'errore in ingresso per ogni unità di output:
 - per calcolare la derivata dell'errore prodotto dei pesi sui layer superiori
 - prodotto dei termini $f'(g(w))$ nella regola di derivazione¹¹ per funzioni composte
 - **layer lineare**: ogni peso aggiornato col prodotto dell'errore in ingresso per il valore in input associato al peso
 - **ogni layer passa il *segnale d'errore*** al layer inferiore
 - regola di composizione → l'errore passato indietro è l'errore nel layer moltiplicato per la derivata della funzione del layer

Ogni layer implementa le due passate attraverso:

- **Output_values(*input*)**
 - restituisce i valori in output per gli *input*
- **Backprop(*error*)**
 - dato l'array *error* di errori delle unità di *output*, aggiorna i pesi e restituisce un array di errori per le unità di *input*

```
class Sigmoid_layer( $n_i$ ) //  $n_i$  num. di input  
procedure Output_values(input) // input array di dim.  $n_i$   
    output[i]  $\leftarrow 1/(1 + \exp(-\text{input}[i]))$  for each  $0 \leq i < n_i$   
    return output
```

```
procedure Backprop(error) // error array di dim.  $n_i$   
    input_error[i]  $\leftarrow \text{output}[i] \cdot (1 - \text{output}[i]) \cdot \text{error}[i]$  for each i  
    return input_error
```

```
class Linear_complete_layer( $n_i, n_o$ )  
    //  $n_i$  num. di input,  $n_o$  num. di output  
    for each  $0 \leq j < n_o$ ,  $0 \leq i \leq n_i$ : Creare  $w_{ji}$ 
```

```
procedure Output_values(input) // input array di dim.  $n_i$   
    definire input[n] pari a 1  
    output[j]  $\leftarrow \sum_{i=0}^n w_{ji} \cdot \text{input}[i]$  for each j  
    return output
```

```
procedure Backprop(error) // error array di dim.  $n_o$   
     $w_{ji} \leftarrow w_{ji} + \eta \cdot \text{input}[i] \cdot \text{error}[j]$  for each i,j, dato  $\eta$   
    input_error[i]  $\leftarrow \sum_j w_{ji} \cdot \text{error}[j]$  for each i  
    return input_error
```

```
procedure Sum_sq_error_layer( $Ys, predicted$ )
// restituisce l'errore iniziale di Backprop()
return [ $Ys[j] - predicted[j]$ ] for each unità di output  $j$ 
```

```
procedure Neural_network_learner( $Xs, Ys, Es, layers, \eta$ )
```

Input

Xs : insieme di feature di input, $Xs = \{X_1, \dots, X_n\}$

Ys : feature obiettivo

Es : insieme di esempi per l'apprendimento

$layers$: sequenza di layer

η : learning rate (dim. del passo in DG)

repeat

for each $e \in Es$ in ordine casuale **do**

$values[i] \leftarrow X_i(e)$ **for each** unità di input i

for each $layer \in layers$ dal più basso al più alto **do**

$values \leftarrow layer.Output_values(values)$

$error \leftarrow \text{Sum_sq_error_layer}(Ys(e), values)$

for each $layer \in layers$ dal più alto al più basso **do**

$error \leftarrow layer.Backprop(error)$

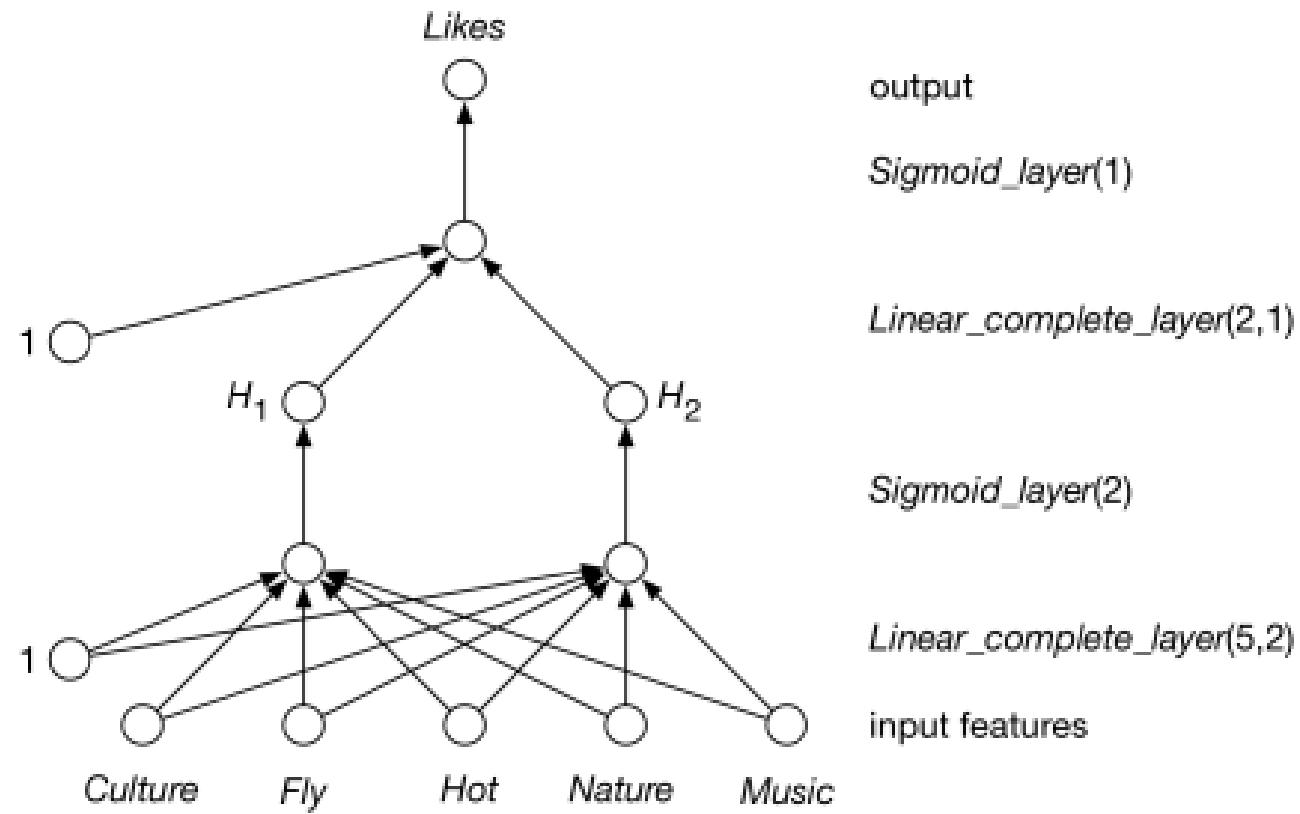
until terminazione

Osservazioni

- variabile *layers* sequenza di layer in cui
 - layer **più basso**: numero di input pari a quello delle feature di input
 - **altri** layer: numero di input pari al numero di output del layer precedente
 - layer **finale**: numero di unità di output pari al numero di feature-target
- **layer lineari**: BP simile a Linear_learner della DG, ma se ne considerano diversi oltre a quelli di attivazione
- per ogni esempio, BP prevede la **simulazione** della rete
 - ad ogni passo, *values* contiene i valori di un solo layer da passare in input al successivo
 - *error* inizializzata con la derivata dell'errore quadratico per gli output
 - valore **passato indietro** attraverso i vari layer
- per ogni peso calcola le derivate con singola passata a ritroso sulla rete

Esempio – Rete per il dataset sulle vacanze

- 5 feature di input e una di output



- **Sequenza di layer:**

$[Linear_complete_layer(5, 2), Sigmoid_layer(2),$
 $Linear_complete_layer(2, 1), Sigmoid_layer(1)]$

- **Un'esecuzione di BP con $\eta = 0.05$, in 10000 passi apprende pesi capaci di predire accuratamente gli esempi di training**
- **Dato un generico esempio e :**

$$H_1(e) = \text{sigmoid}(-2.0 \cdot \text{Culture}(e) - 4.4 \cdot \text{Fly}(e) + \\ + 2.5 \cdot \text{Hot}(e) + 2.4 \cdot \text{Music}(e) - 6.1 \cdot \text{Nature}(e) + 1.6)$$

$$H_2(e) = \text{sigmoid}(-0.7 \cdot \text{Culture}(e) + 3.0 \cdot \text{Fly}(e) + \\ + 5.8 \cdot \text{Hot}(e) + 2.0 \cdot \text{Music}(e) - 1.7 \cdot \text{Nature}(e) - 5.0)$$

$$\widehat{\text{Likes}}(e) = \text{sigmoid}(-8.5 \cdot H_1(e) - 8.8 \cdot H_2(e) + 4.4)$$

- **Con più esecuzioni si possono avere pesi anche molto diversi**

Osservazione: apprendimento *sub-simbolico*

Contrasto apparente con l'ipotesi di sistema di simboli fisico intrinseco

- la rete codifica un significato che si può solo intravedere dai valori delle unità nascoste
 - ma a tali unità non è associato un *significato preciso*
- in alcuni casi si può interpretare cosa rappresentino ed esprimerlo in forma concisa

cfr. XAI: eXplainable AI

ESTENSIONI

Layer multipli → modellazione gerarchica → **deep learning [9]**

- **RN convolutiva** (*convolutional neural network*, CNN)
 - feature locali, specializzata nel riconoscere immagini / visione
- **RN ricorrente** (*recurrent neural network*, RNN)
 - oltre il feed-forward, serie temporali

Reti per *applicazioni reali*:

- da 10 a 20 layer con $\sim 10^{10}$ pesi
- richiedono ore/giorni di addestramento su macchine con migliaia di core

Esercizi: esercitarsi su diversi dataset e problemi di classificazione/regressione usando <https://playground.tensorflow.org/>

- o altre librerie sulle RN riusabili ai fini del progetto

Estensioni dei Modelli Lineari

I modelli d'apprendimento visti forniscono le basi per altre tecniche:

- ogni funzione discreta rappresentabile con alberi di decisione/regressione
 - necessariamente anche molto complessi
- i classificatori lineari rappresentano classi di funzioni limitate, ma le RN, con strati di funzioni lineari + strati con funzioni di attivazione *non lineari*, possono approssimare funzioni più complesse

Estensione:

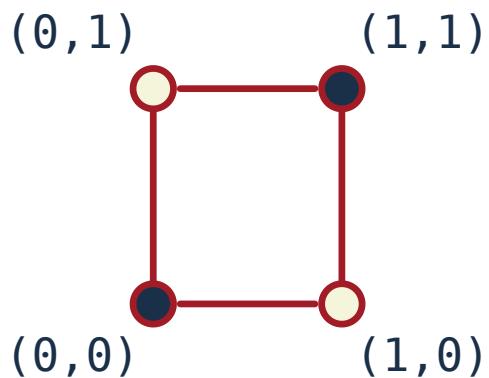
funzione non lineare degli input originari → input di una funzione lineare

- nuove feature → aumenta la *dimensionalità* dello spazio e funzioni/problemi che non lo erano *diventano lineari*
(o almeno *linearmente separabili*)

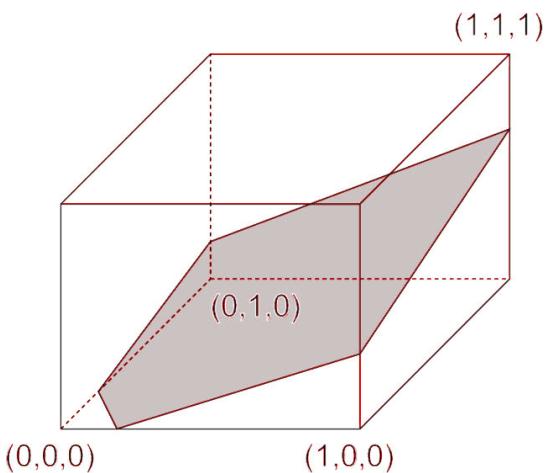
Esempio – xor linearmente separabile nello spazio 3d con feature le x_1 , x_2 e x_1x_2 :

- $x_1x_2 = 1$ (*vero*) quando $x_1 = x_2 = 1$
- $x = x_1, y = x_2$ e $z = x_1x_2$
 - e.g. $(1, 1)$ mappato su $(1, 1, 1)$
 - *separazione* con un piano

2D



3D



Da [10]

Una funzione **kernel**¹² produce **nuove feature** a partire da quelle di input

[6, 7, 8, 9]

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

- generalizza il **prodotto vettoriale**, interpretabile con misura di **similarità**
 - ad es., prodotto di feature aggiuntivo/sostitutivo per quelle iniziali
- moltissimi kernel disponibili per diversi spazi di dati, anche non vettoriali
 - stringhe, liste, alberi, grafi, testi, distribuzioni di probabilità,...
- nuovo spazio di feature (**embedding space**) utile alla separabilità lineare
 - ad es., a x aggiungere x^2 e x^3 , e cercare il miglior polinomio di terzo grado

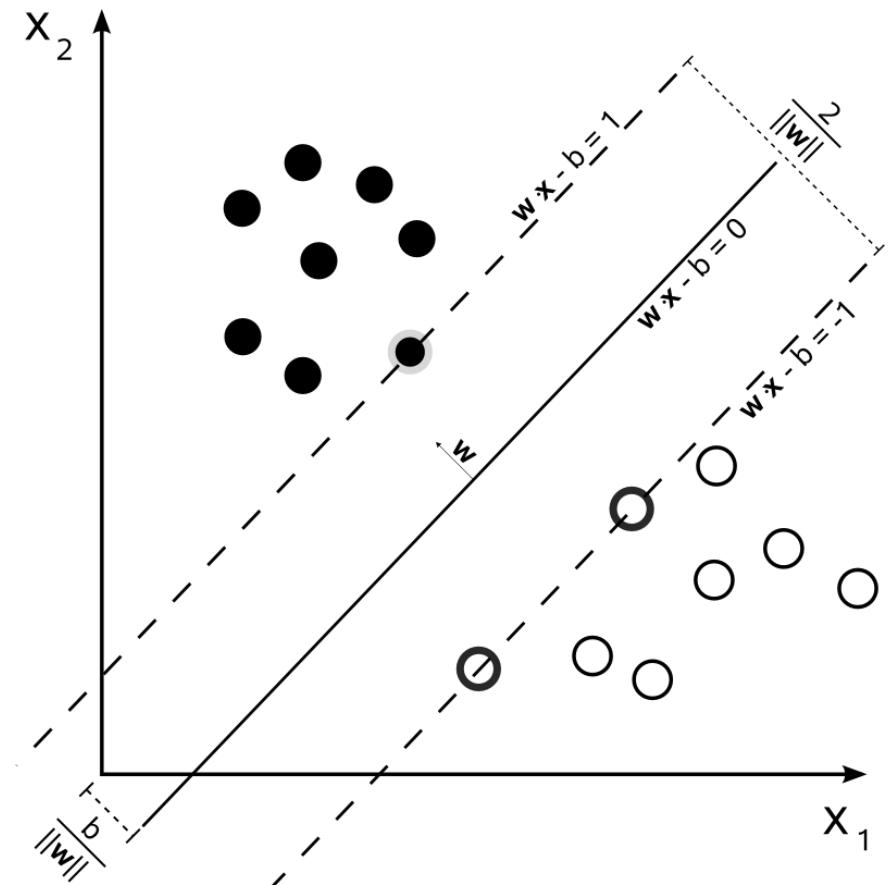


l'estensione dello spazio delle feature rende il problema del **sovradattamento** più cogente¹³

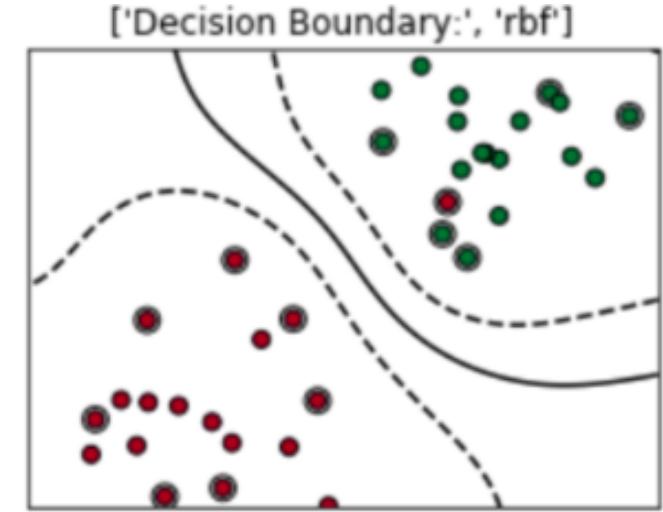
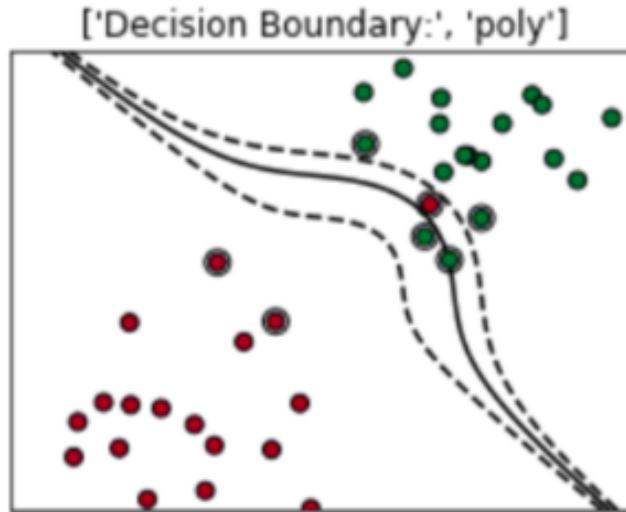
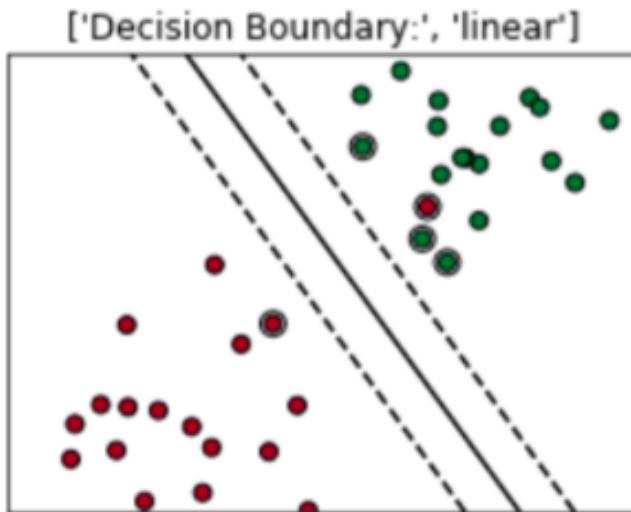
SUPPORT VECTOR MACHINE (SVM)

Classificatore¹⁴ basato su *kernel* [KM]:

- definisce un iperpiano, *superficie di decisione* ottimale, dividendo positivi da negativi nello spazio di feature che dipende dal kernel (ossia da ϕ)
 - *margine*: distanza minima degli esempi dall'iperpiano
 - la SVM trova l'iperpiano di *massimo margine*
 - esempi sul margine: vettori di *supporto*
- soluzione (parametri dell'iperpiano) calcolata sulla base di prodotti (kernel) applicati a tali vettori
 - evita il sovradattamento: modello determinato da pochi vettori



Spazi determinati da kernel (ϕ) diversi



Live demo @ [Stanford](#)

ULTERIORI ESTENSIONI

- **Reti neurali:** ammettono input alla funzione lineare (appiattita) che sono a loro volta funzioni lineari appiattite con coefficienti da determinare:
 - con diversi layer siffatti si possono rappresentare *funzioni più complesse*
- **Alberi di regressione:** alberi con funzioni lineari (appiattite) alle foglie
 - altra rappresentazione non lineare: possono rappresentare *approssimazioni lineari a pezzi*
 - estensioni: reti neurali o altri classificatori sulle foglie
 - per predizioni su un nuovo esempio:
 - lo si instrada giù per l'albero dalla radice a una foglia
 - si usa il modello contenuto nella foglia

MODELLI COMPOSITI

Ensemble Learning

In generale, nell'**ensemble learning**, si **combinano** le predizioni di un dato numero di modelli ciascuno appreso da learner *di base*:

- ad es. random forest
 - alberi di decisione come base
 - predizioni finali come media o voto
- Modelli di **ensemble learning**: *bagging, boosting, stacking, ...*
 - usare un certo numero di modelli-base addestrati sui dati
 - combinati attraverso un meccanismo di voto o di mediazione

BAGGING: RANDOM FOREST

Random Forest: modello composto da più alberi di decisione

- **idea:** si addestra un certo numero di alberi su parte del dataset
 - per ogni esempio da classificare ognuno fa una predizione
 - si aggregano le predizioni per formare la predizione finale per l'esempio
- efficace in base alla **diversità** degli alberi generati: diverse predizioni
 - per ogni albero: sottoinsieme **casuale** di feature
 - ad es. $1/n$ delle feature
 - feature per i test nei nodi scelte in un **insieme ristretto**
 - insieme variabile per albero / per nodo
 - **bagging:** diversi sottoinsiemi di esempi di training
 - dati m esempi di training ogni albero ne usa un piccolo numero
 - sottoinsieme di m esempi estratti casualmente (con rimpiazzo)
 - in ognuno, esempi assenti o duplicati → ~63.2% degli esempi originari
- **predizione** finale mediando: e.g. **classe** di maggioranza sulle predizioni
 - **voto** = classe più probabile per il singolo albero

BOOSTING

Modelli in **sequenza** costruiti imparando dagli errori dei precedenti:

- stesso tipo di **learner di base** (ma anche differenziati)
 - ad es. alberi di decisione o funzioni lineari appiattite
- da adattare agli esempi sui quali il modello precedente commette errori
 - esempi **ordinati** in base all'errore
- predizione **finale** aggregando le predizioni dei modelli prodotti nelle varie iterate
 - somma, media pesata, moda

Learner di base: **deboli**, non necessariamente ottimali – **weak learner**

- devono giusto superare un classificatore casuale
- l'ensemble avrà prestazioni migliori rispetto a ciascuno di essi

Fra i più noti **ADABOOST** (**ADaptive BOOSTing**¹⁵)

- funzione d'errore: $E(\mathbf{x}) = e^{-y\hat{Y}(\mathbf{x})}$

procedure AdaBoost(E_s, T)

Input

$E_s = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, n\}$ esempi

$T \in \mathbb{N}$ numero iterate

Output

\hat{Y} modello finale (strong)

for $t = 1$ **to** T

for $i = 1$ **to** n **do** $w_i^{(t)} \leftarrow 1/n$ // inizializzazione pesi

Trovare $h^{(t)}(\mathbf{x}) : \mathbf{x} \mapsto \{-1, +1\}$ // modello weak

che minimizza $\varepsilon_{(t)} = \sum_{i|h^{(t)}(\mathbf{x}_i) \neq y_i} w_i^{(t)}$

$\alpha_{(t)} \leftarrow \frac{1}{2} \ln(1 - \varepsilon_{(t)}) / \varepsilon_{(t)}$

$\hat{Y}^{(t)} \leftarrow \hat{Y}^{(t-1)} + \alpha_{(t)} h^{(t)}$ // aggiunta all'ensemble

$S \leftarrow 0$

for $i = 1$ **to** n **do** // aggiornamento

$w_i^{(t+1)} \leftarrow w_i^{(t)} e^{-y_i \alpha_{(t)} h^{(t)}(\mathbf{x}_i)}$

$S \leftarrow S + w_i^{(t+1)}$

for $i = 1$ **to** n **do** $w_i^{(t+1)} \leftarrow w_i^{(t+1)} / S$ // ri-normalizzazione

return $\hat{Y}^{(T)}$

REGRESSIONE VIA FUNCTIONAL GRADIENT BOOSTING

- ***predizione finale*** in funzione degli input:

$$p_0(X) + d_1(X) + \cdots + d_k(X)$$

- $p_0(X)$ predizione iniziale, e.g. la media
- d_i differenza dalla predizione precedente

- ***i*-esima predizione:** $p_i(X) = p_0(X) + d_1(X) + \cdots + d_i(X) = p_{i-1}(X) + d_i(X)$
- **d_i costruito dal weak learner minimizzando l'errore di p_i (fissato p_{i-1}):**

$$\sum_e error(Y_i(e) - p_i(e)) = \sum_e error(Y_i(e) - p_{i-1}(e) - d_i(e))$$

- impara $d_i(e)$ per ottimizzare $Y_i(e) - p_{i-1}(e)$
 - come dataset con output modificati sottraendo le predizioni precedenti
 - ogni learner tende a correggere gli errori precedenti

procedure Boosting_learner(X_s, Y, E_s, L, k)

Input

X_s : insieme di feature di input

Y : feature obiettivo

E_s : insieme di esempi di training

L : learner di base

k : numero di componenti dell'ensemble

Output

funzione per fare predizioni sugli esempi

$$media \leftarrow \sum_{e \in E_s} \frac{Y(e)}{|E_s|}$$

define $p_0(e) = media$

for $i = 1$ **to** k **do**

$$E_i \leftarrow \{\langle X_s(e), Y(e) - p_{i-1}(e) \rangle \mid e \in E_s\}$$

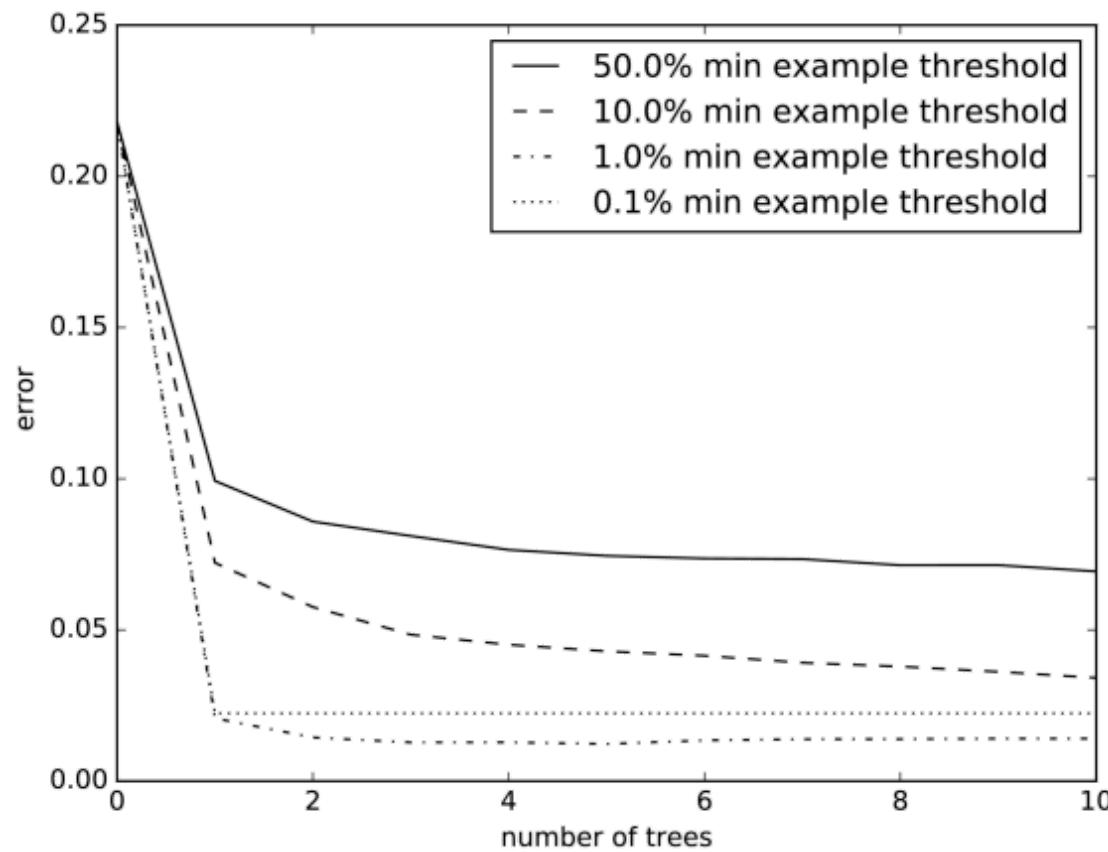
$$d_i \leftarrow L(E_i)$$

$$\text{definire } p_i(e) = p_{i-1}(e) + d_i(e)$$

return p_k

- p_i funzione di predizione su esempi
- E_i nuovo insieme di esempi, generabile anche on demand
 - per ogni $e \in E_s$: predizione più recente, $p_{i-1}(e)$, sottratta da $Y(e)$
- d_i calcolata applicando il learner-base a E_i

Grafico dell'errore quadratico all'aumentare del numero di alberi



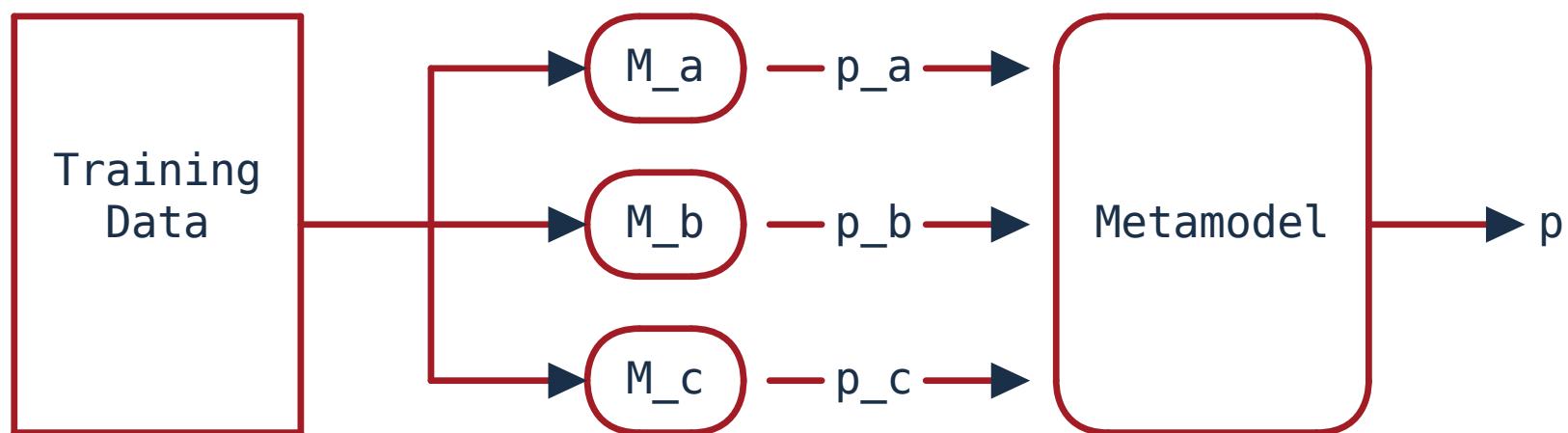
- linee diverse al variare delle soglie per il minimo tasso di esempi rispetto al totale per il partizionamento
- il boosting rende gli alberi con soglia all'1% migliori di quelli al 0.1%

STACKING

Ensemble model costituito da:

[6, 7, 8]

- **modelli-base** diversi
 - e.g. alberi, mod. lineari, SVM,...
- **meta-learner** che apprende come aggregare al meglio le decisioni



CASE-BASED REASONING

In alternativa alla costruzione di un modello,
rappresentazione compatta dei dati, nel **case-based reasoning**:

- esempi di training – **casi** – immagazzinati per essere poi ritrovati nella soluzione di problemi
 - predizione su nuovo esempio:
ritrovamento di casi **simili** per determinare il valore della **Y**
 - carico di lavoro spostato sul **query time** (predizione)
 - poco lavoro offline (memorizzazione esempi)
- problemi di **classificazione** e **regressione**

K-NEAREST NEIGHBORS

- dato un nuovo esempio, per predirne il valore target, si cercano i k esempi più simili memorizzati
 - **predizione:** moda, media o interpolazione dei valori-target dei k esempi
 - anche con un peso che dipende dalla similarità
- richiede una **metrica** per determinare la similarità degli esempi
 - per ogni feature: rappresentazione numerica $X_i(e)$ del valore di X_i per e
 - dati e_1 ed e_2 , si misura: $X_i(e_1) - X_i(e_2)$
 - tipicamente usata nella **distanza Euclidea**
 - scale diverse tra feature da omogeneizzare
 - parametri non-negativi w_i per pesare l'importanza di ogni X_i :

$$d(e_1, e_2) = \sqrt{\sum_{i=1}^n w_i \cdot [X_i(e_1) - X_i(e_2)]^2}$$

- forniti in input o appresi a parte,
e.g. minimizzando un errore sulle predizioni via CV **leave-one-out**

Esempio – Si consideri il dataset iniziale

- per classificare e_{20} , con $Author = unknown$, $Thread = followup$,
 $Length = short$, $Where_read = at_home$, si cercano casi simili:
 - e_{11} ha corrispondenza esatta quindi, come per e_{11} , si può predire $skips$
- per classificare e_{19} con $Author = unknown$, $Thread = new$,
 $Length = long$, $Where_read = work$
 - non ci sono corrispondenze esatte, ma:
 - e_2 , e_8 e e_{18} concordano su $Author$, $Thread$ e $Where_read$
 - e_{10} e e_{12} concordano su $Thread$, $Length$ e $Where_read$
 - e_3 concorda su $Author$, $Length$ e $Where_read$
 - predizione: e_2 , e_8 e e_{18} predicono $reads$, gli altri $skips$
 - albero di decisione: $Length$ feature migliore → andrebbero ignorati e_2 , e_8 ed e_{18}
 - modello lineare con sigmoide:
dai parametri appresi in un esempio precedente si dovrebbe predire $skips$
 - CBR: determinante l'importanza delle diverse feature (pesi)

Problema: accesso ai casi rilevanti

- **kd-tree** indice per esempi di training,
ritrovamento veloce per **similarità**
 - come un albero di decisione con test basati sulle X_i
 - foglie = partizioni degli esempi simili
 - **costruzione**: trovare X_i che partiziona più equamente gli esempi del nodo
 - ripetendo ricorsivamente per la costruzione di ogni sotto-albero/partizione
 - criterio di **stop**: pochi esempi alle foglie
 - **ricerca**: sulla base dell'esempio in input come per gli alberi di decisione
 - corrispondenze esatte fino alle foglie solo per le X_i nei test dei nodi interni attraversati
 - ma rispetto ad altre feature casi anche molto differenti
 - stesso albero anche per esempi con una sola feature diversa da quelle testate:
per nuovi casi si prevede un ramo aggiuntivo per un valore diverso

CBR su problemi complessi: problemi di planning o casi legali

- scelta + eventuale modifica (editing) dei casi
- iterando i passi seguenti:
 - **Ritrovamento:** dato un nuovo caso, ritrovare casi simili nella KB
 - **Riuso:** adattare i casi ritrovati al nuovo caso
 - **Revisione:** valutare la soluzione e rivederla in base alla sua qualità
 - **Ritenzione:** decidere se memorizzare il nuovo caso adattato nella KB

Se il caso è conforme alla situazione corrente se ne adotta la soluzione, altrimenti lo si raffina:

- revisione basata anche su tecniche diverse:
 - es. soluzione proposta come punto di partenza per una ricerca
 - es. adattamento interattivo con intervento umano
- nuovo caso e soluzione da memorizzare se può tornare utile in futuro

Esempio – *Help desk* clienti: diagnostica problemi con i computer

- fornita la descrizione del problema, si ritrovano i casi più simili
- si possono raccomandare all'utente le soluzioni trovate,
anche adattate al caso specifico
 - es. cambiare l'azione consigliata in base al tipo di SW sulla macchina,
il tipo di connessione di rete o il modello della stampante collegata
- se un caso adattato risolve, è aggiunto alla KB per essere riusato
 - i casi più comuni alla fine saranno nella KB
- altrimenti
 - provare altri metodi di soluzione
 - e.g. adattando altri casi o richiedendo l'intervento umano
 - risolto il problema, si aggiunge la soluzione alla KB
- possibile rimozione di casi simili ridondanti
 - *C ridondante* se esistono C_i tali che ogni caso con *C* come più prossimo,
in sua assenza li userebbe, ricevendone la stessa soluzione

APPRENDIMENTO COME RICERCA: RAFFINAMENTO DI IPOTESI ↵

Apprendimento come *ricerca* delle *ipotesi coerenti* con i dati

[6, 8]

- i.e. che non contraddicono gli esempi osservati
- invece di lavorare su una sola ipotesi,
trovare una descrizione per l'insieme dei modelli coerenti con i dati

Assunzioni

- *singola* feature-oggettivo Y booleana
- ipotesi per *predizioni definite* (i.e. non probabilistiche)
- non c'è *rumore* nei dati

Scopo: cercare ipotesi che classifichino correttamente gli esempi

- *Ipotesi:* proposizione
 - *atomi* = assegnazioni alle feature di input

Esempio – Predire la lettura di articoli in base a keyword

- dati gli esempi:

articolo	<i>Crime</i>	<i>Academic</i>	<i>Local</i>	<i>Music</i>	<i>Reads</i>
a_1	true	false	false	true	true
a_2	true	false	false	false	true
a_3	false	true	false	false	false
a_4	false	false	true	false	false
a_5	true	true	false	false	true

- scopo: caratterizzare gli articoli che verranno letti
 - *Reads* feature-obiettivo
 - per classificare anche esempi futuri, trovare definizioni come:

$$\widehat{\text{Reads}}(e) \leftrightarrow \text{Crime}(e) \wedge (\neg \text{Academic}(e) \vee \neg \text{Music}(e))$$

Insiemi di interesse per l'apprendimento:

- I : spazio delle istanze, insieme di tutti i possibili esempi
- \mathcal{H} : spazio delle ipotesi, insieme di funzioni booleane sulle feature di input
- E : insieme degli esempi di training, $E \subseteq I$
 - con valori per le feature di input e target

Se $h \in \mathcal{H}$ e $i \in I$: $h(i)$ valore predetto da h per i

Esempio – Nel caso visto in precedenza

- I insieme dei $2^5 = 32$ possibili esempi
 - uno per ogni combinazione dei valori delle 5 feature
- \mathcal{H} contiene tutte le combinazioni booleane delle feature di input
 - o restrizioni, come le sole congiunzioni oppure le proposizioni definite in termini di meno di d feature
- $E = \{a_1, a_2, a_3, a_4, a_5\}$

Per classificare gli esempi futuri serve un $bias$, qui imposto da \mathcal{H}

COERENZA

Ipotesi h coerente (*consistent*) con E sse
per ogni esempio in E il valore h predice il valore di Y :

$$\forall e \in E: h(e) = Y(e)$$

Esempio – Considerato lo stesso dataset precedente,
sia \mathcal{H} l'insieme delle congiunzioni di letterali

- un'ipotesi $h \in \mathcal{H}$ coerente con $\{a_1\}$ è

$$h: \widehat{\text{Reads}}(e) \leftrightarrow \neg \text{academic}(e) \wedge \text{music}(e)$$

“si legge l'articolo sse non è di carattere accademico e riguarda la musica”

- non è l'ipotesi finale cercata
essendo in contraddizione già con $\{a_1, a_2\} \subset E$

Problema: trovare $\mathcal{H}_{VS} \subseteq \mathcal{H}$ delle ipotesi coerenti con tutti gli $e \in E$

- Soluzione poco efficiente: enumerazione + selezione
- **Idea:** imporre una *struttura* su \mathcal{H} per velocizzare la ricerca

h_1 più generale di h_2 se h_1 implicato da h_2

- in tal caso: h_2 più specifica di h_1 : $h_2 \preceq h_1$
 - **NB** ogni h è sia più specifica sia più generale di se stessa: $h \preceq h \wedge h \preceq h$
- ipotesi più generale di \mathcal{H} : *true*
- ipotesi più specifica di \mathcal{H} : *false*
- quindi, per esempio:
 - $\neg academic \wedge music \preceq \neg academic$
 - $\neg academic \wedge music \preceq music$

SPAZIO DELLE VERSIONI

La relazione \preceq costituisce un *ordinamento parziale*¹⁶ su \mathcal{H}

- sfruttato nell'algoritmo di ricerca che segue

Spazio delle Versioni o *Version Space* (VS):
sottoinsieme delle ipotesi di \mathcal{H} coerenti con E

Per determinare completamente il VS:

- G , **confine generale** (*general boundary*) del VS
 - insieme delle ipotesi massimali per \preceq
 - nessun'altra ipotesi del VS è più generale
- S , **confine specifico** (*specific boundary*) del VS
 - insieme delle ipotesi minimali per \preceq

Proposizione – Il VS è l'insieme di $h \in \mathcal{H}$ tali che h sia più generale di una in S e più specifica di una in G

Candidate Elimination

Costruzione incrementale del VS, dati \mathcal{H} ed E :

[6]

- esempi elaborati uno alla volta
- ognuno può provocare una riduzione del VS con la rimozione delle ipotesi incoerenti con E
 - aggiornando S e G

Si può definire anche per spazi di ipotesi più espressive,
e.g. clausole di Horn del I ordine,
avendo definito un'opportuna relazione d'ordine parziale (cfr. ILP [6, 8])

procedure Candidate_elimination_learner(X_s, Y, E_s, \mathcal{H})

Input

X_s : feature di input, $X_s = \{X_1, \dots, X_n\}$

Y : feature-obiettivo booleana

E_s : esempi di training

\mathcal{H} : spazio delle ipotesi

Output

confine generale $G \subseteq \mathcal{H}$

confine specifico $S \subseteq \mathcal{H}$ coerenti con E_s

Local

G : sottoinsieme di \mathcal{H}

S : sottoinsieme di \mathcal{H}

Let $G = \{true\}$, $S = \{false\}$

for each $e \in Es$ **do**

if $Y(e) = true$ **then**

1. Rimuovere ogni $g \in G$ per la quale e negativo;
2. Sostituire ogni $s \in S$ per la quale e negativo con le sue generalizzazioni minimali che lo classificano come positivo ma risultino più specifiche di una $g \in G$;
3. Rimuovere da S ipotesi non massimali

else // e negativo

1. Rimuovere ogni $s \in S$ per la quale e positivo;
2. Sostituire ogni $g \in G$ che classifichi e come positivo con le sue specializzazioni minimali che lo classifichino come negativo risultando più generali di una $s \in S$;
3. Rimuovere da G eventuali ipotesi non minimali

Esempio – Applicando l'algoritmo al dataset precedente,
con \mathcal{H} costituito dalle congiunzioni di letterali

- inizialmente:

$G_0 = \{\text{true}\}$ (si legge tutto) e $S_0 = \{\text{false}\}$ (non si legge nulla)

- true : congiunzione vuota (no vincoli)
- false : congiunzione contenente (almeno) un atomo e la sua negazione

- dopo a_1 : $G_1 = \{\text{true}\}$ e $S_1 = \{\text{crime} \wedge \neg\text{academic} \wedge \neg\text{local} \wedge \text{music}\}$

- ipotesi più generale: si legge tutto;
- ipotesi più specifica: si leggono articoli con le stesse feature di a_1

- dopo anche a_2 : $G_2 = \{\text{true}\}$ e $S_2 = \{\text{crime} \wedge \neg\text{academic} \wedge \neg\text{local}\}$

- dato che a_1 e a_2 , positivi, non concordano su music ,
questa viene giudicata irrilevante

- dopo a_3 : $G_3 = \{crime, \neg academic\}$ e $S_3 = S_2$
 - due ipotesi generali massimali:
 - si legge tutto ciò che sia di tipo *crime*
 - si legge tutto ciò che non sia a carattere accademico
- dopo a_4 : $G_4 = \{crime, \neg academic \wedge \neg local\}$ e $S_4 = S_3$
- dopo i cinque articoli: $G_5 = \{crime\}$, $S_5 = \{crime \wedge \neg local\}$
 - rimangono solo 2 ipotesi che si distinguono solo rispetto alla predizione di esempi sulla base della verità di *crime* \wedge *local*
 - se il concetto-obiettivo può essere rappresentato con una congiunzione, solo un esempio con *crime* \wedge *local* vera cambierà G o S

Il VS potrà essere usato per predizioni su nuovi esempi

BIAS NELL'APPRENDIMENTO DEL VERSION-SPACE ↪

Il **bias** serve a generalizzare, andando oltre gli esempi di training

- Nel **version-space**, **language bias** o **restriction bias**
 - perché ottenuto restringendo lo spazio delle ipotesi
 - un nuovo esempio con *crime* falso e *music* vero sarà classificato come falso anche se questo non era stato osservato tra gli esempi di training
 - la restrizione alle sole congiunzioni basta a far sì che si possa predire la classificazione
- Bias negli **alberi di decisione**
 - possono rappresentare qualunque funzione booleana
 - **preference bias**: funzioni semplici corrispondenti agli alberi più piccoli favorite rispetto a quelle più complesse
 - **search bias**: l'albero restituito dipende dalla strategia di ricerca

- CANDIDATE ELIMINATION sembra non avere bias oltre a quello sul linguaggio di \mathcal{H}
 - facile che il VS collassi all'insieme vuoto (concetto cercato non in \mathcal{H})
 - ad es. con un articolo in cui *crime* sia falso e *music* vero
 - l'algoritmo non ammette *esempi rumorosi*:
 - basta un solo esempio con classificazione errata per compromettere l'intero risultato
- \mathcal{H} bias-free quando contiene tutte le funzioni booleane
 - G conterrà sempre un solo concetto:
 - per il quale tutti i negativi sono stati considerati (in fase di training) e ogni altro esempio sarà positivo
 - Analogamente, S conterrà un solo concetto
 - per il quale tutti gli esempi non considerati saranno negativi
- Il VS non sa concludere nulla su esempi non visti
 - non sa generalizzare
 - senza un (language o preference) bias non si avrà alcuna generalizzazione

RIFERIMENTI

Bibliografia

- [1] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. 2nd Ed. [Ch.7]
- [2] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press
- [3] S. J. Russell, P. Norvig: *Artificial Intelligence* Pearson. 4th Ed. (ch.19) - cfr. anche ed. Italiana
- [4] J. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations* Brooks Cole/Cengage
- [5] F. Rosenblatt: *The perceptron: a probabilistic model for information storage and organization in the brain* Psychological Review 65 (6), pp. 386-408 (1958)
- [6] T. Mitchell: *Machine Learning*. McGraw Hill
- [7] T. Hastie, R. Tibshirani, J. Friedman: *The Elements of Statistical Learning* Springer. 2nd ed [ESL]
- [8] P. Flach: *Machine Learning*. Cambridge University Press
- [9] D.J.C. MacKay: *Information Theory, Inference, and Learning Algorithms* Cambridge University Press [ITPRN]
- [10] I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*. MIT Press [DLBook]
- [11] J.M. Bishop: *History and Philosophy of Neural Networks*. In: *Computational Intelligence*, Ch:2. Eolss 2015 [HPNN]

[KM] <http://www.kernel-machines.org> <https://kernelmethods.blogs.bristol.ac.uk/>
 [DLBook] <http://www.deeplearningbook.org>
 [ESL] <https://web.stanford.edu/~hastie/ElemStatLearn/>
 [ITPRN] <http://www.inference.org.uk/itprnn/book.html>
 [HPNN] https://www.researchgate.net/publication/271841595_HISTORY_AND PHILOSOPHY_OF_NEURAL_NETWORKS

NOTE

¹ delta di Kronecker
² norme e distanze

³ uno dei valori che compaiono più frequentemente; in caso di più mode si può scegliere una qualunque

⁴ numero centrale nella lista ordinata dei valori, preceduto e seguito da almeno la metà dei valori: ad es., per {2, 2, 4, 7, 7, 15}, ogni numero tra 4 e 7 è mediano
⁵ generalizzabili ai casi n-ari (feature discrete) e multivariati (test su più feature di input)

⁶ Funzione gradino di Heaviside

⁷ Regressione verso la media

⁸ Bias-variance tradeoff

[◀] consigliata la lettura

[versione] 29/11/2022, 23:45:16

¹⁰ Smoothing Additivo su <en.wikipedia>

¹¹ Su <it.Wikipedia> e <en.Wikipedia>

regole utili:

regola lineare: $\frac{\partial}{\partial w} aw + b = a$

regola di composizione: $\frac{\partial}{\partial w} f(g(w)) = f'(g(w)) \cdot \frac{\partial}{\partial w} g(w)$

¹² Si veda [KM] e, inoltre, *kernel trick* su <wikipedia>, <kernel> e <SVM>

¹³ *Curse of dimensionality* su <Quora>, <WP>, <Medium-1> <Medium-2>

¹⁴ SVM in <en.wikipedia>, <it.wikipedia>

¹⁵ ADaptive BOOSTing su <en.wikipedia>

¹⁶ Su <en.wikipedia>

Figure tratte da [1] salvo diversa indicazione

formatted by <Markdeep 1.14>