

7. Apprendimento Supervisionato

Dispensa ICon

versione: 28/11/2024, 12:38

Apprendimento: Problematiche · Fondamenti · Modelli-Base · Sovradattamento · Modelli Compositi

L'**apprendimento** è la capacità di sfruttare l'*esperienza* per migliorare il *comportamento* nello svolgimento di specifici compiti:

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

Informalmente si può anche dire che l'obiettivo dell'apprendimento è comprendere il mondo attraverso i **dati**. Occorre modellare non tanto i dati quanto il mondo che li *genera* per consentire il miglioramento delle decisioni / azioni del sistema intelligente. La conoscenza del passato non è fine a se stessa ma dev'essere utile per il futuro. Ad esempio, dati storici bancari su una persona, il riconoscimento (dei volti) delle persone nella guida automatica, o dati raccolti da smartwatch saranno utili ad apprendere modelli di predizione di decisioni future sulla concessione di mutui la posizione, sulla guida e l'apertura automatica delle porte o sulla salute di una persona in casi di emergenza.

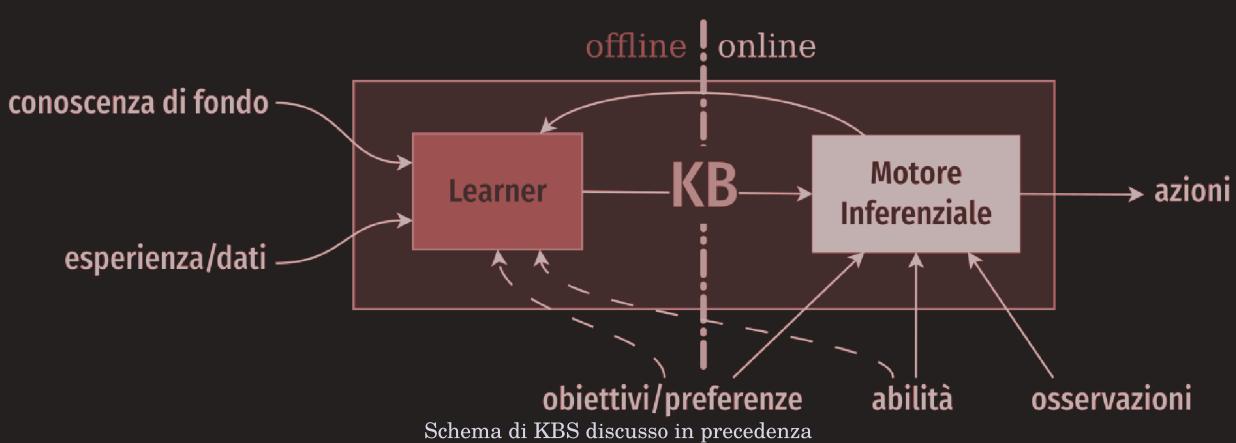
Nel seguito verranno trattate problematiche specifiche legate alla predizione tramite apprendimento supervisionato: dato un insieme di *esempi* di training costituito da coppie input-output, si dovrà predire l'*output* per nuovi esempi dato il solo input. Gli esempi si chiamano anche *campioni* mentre l'output da predire si chiama anche o *target*. Si vedranno nel seguito algoritmi di base e modelli più sofisticati.

1 Apprendimento: Problematiche

In un *problema di apprendimento automatico* si distinguono:

- il **task**, ovvero il *comportamento* o *compito* nel quale migliorare;
- i **dati**, ossia le *esperienze* usate per migliorare le prestazioni nello svolgimento del compito, di solito disponibili come sequenza di *esempi*;
- la **misura del miglioramento**: occorre stabilire *come* e *cosa* misurare, come ad esempio l'incremento di abilità, l'efficacia, l'efficienza nella soluzione del problema.

Ai fini della predizione, si considereranno la *conoscenza di fondo* (*background knowledge*, BK) e i *dati*, ossia l'*esperienza* pregressa, tipicamente in forma di esempi, ossia coppie **input-output**, per creare una *rappresentazione interna*, un **modello**, che potrà costituire parte della KB (cfr. figura seguente), da usare per future *predizioni* (su azioni/decisioni):



Nel seguito sono elencate le diverse problematiche legate alla soluzione dei problemi di apprendimento automatico:

- **Task:** il compito per cui si devono acquisire dati / esperienza. Nell'**apprendimento supervisionato**, dato un insieme di **esempi di training** descritti in termini di *feature* (caratteristiche, attributi) di *input* e *target* (obiettivo), si deve imparare a predire (online) il valore ignoto delle feature target per nuove osservazioni. Nei problemi di *classificazione* il target è discreto mentre in quelli di *regressione* il target continuo. Quando la feature target ha una struttura si parla di *predizione strutturata*. Altri problemi sono legati a forme di *apprendimento non-supervisionato*, in cui si cercano regolarità o *pattern*, senza avere a disposizione dati con feature target esplicite. Nell'*apprendimento relazionale* si costruiscono modelli per *rappresentazioni più espressive* dei dati, quindi relazioni, concetti / programmi logici (ILP), grafi. Nell'*apprendimento per rinforzo* l'obiettivo è pianificare comportamenti ovvero piani su azioni da svolgere. Nei problemi di *ranking* si impara a ordinare le istanze, ad esempio gli item trattati dai sistemi di raccomandazione. Infine nell'*apprendimento analitico* l'obiettivo è imparare a ragionare ovvero a prendere decisioni più *velocemente*.
- **Feedback:** il tipo di feedback che si riceve dipende dal task di apprendimento. Nei task supervisionati, gli *esempi di training* forniti da un esperto/addestratore, contengono il valore-target corretto per ognuno di essi. Nei task non-supervisionati non è specificato un output desiderato, si hanno solo *osservazioni* con le sole feature di input avvalorate, quindi ci si basa sulla valutazione di funzioni che misurano le prestazioni già menzionate. Nei task per rinforzo si hanno *ricompense* e *punizioni* che danno un valore alle diverse sequenze di azioni svolte in base al modello appreso.
- **Misura delle prestazioni**, ossia del miglioramento dei modelli su *nuovi* dati. Tipicamente nella *valutazione* si dividono i dati in due gruppi, *training* e *test*: costruito un *modello (ipotesi)* usando gli esempi di training, se ne misura l'*accuratezza predittiva* su *nuovi* esempi, quelli di test. Si ottiene una misura *approssimata* delle prestazioni, dato l'insieme limitato di dati, che va *ripetuta* al variare delle diverse suddivisioni del campione di dati. Nel caso supervisionato, le ipotesi, indotte da *campioni* della popolazione, devono tendere a *generalizzare* il training set, senza farsi fuorviare da sue specificità, altrimenti risulterebbero scarsamente *predittive* rispetto a nuovi dati. Ad esempio in un *problema di classificazione binario*, in cui si decide l'appartenenza a una classe, si potrebbe formulare facilmente un'ipotesi *h_P* che decide sempre la predizione positiva, tranne che per i soli negativi del training set e una ipotesi *h_N* che decida sempre per la predizione negativa, tranne che per i soli positivi del training set; tali ipotesi sono modelli *estremi*: per entrambe si avrebbe il 100% di *accuratezza* sui dati di training ma risulterebbero in *totale disaccordo* sul resto degli esempi. Le ipotesi vanno sempre valutate su nuovi dati per determinare la loro generalità.

- **Bias:** tendenza a preferire certe ipotesi rispetto ad altre. Senza bias, come si può stabilire che un'ipotesi sia migliore di un'altra? Nell'esempio precedente le ipotesi h_N o h_P sono entrambe accurate al 100% sui dati di training: quale preferire? Serve un *fattore esterno* che prescinda anche dai dati, in grado di ottimizzare le predizioni su nuovi esempi (h_P e h_N saranno in disaccordo su tutti gli esempi futuri). La *scelta* del bias è problema empirico da risolvere attraverso la pratica. Tipicamente, ci si basa su misure di *complessità* delle ipotesi (in seguito si tratterà il *rasoio di Occam*).
- **Rappresentazione**, ossia la *codifica interna* dell'esperienza, la forma dell'ipotesi / del modello. La più semplice è la forma *grezza*, in cui si memorizzano le esperienze stesse, gli esempi. Una forma *compatta* è più comune perché permette la generalizzazione dei dati. La scelta fra le rappresentazioni possibili dei modelli viene detta **representation bias** ed è correlata alla scelta fra i diversi modelli, o **preference bias**. Come *linea-guida*, si sa che optando per rappresentazioni più ricche sarà più facile esprimere ipotesi generali (più predittive) ma che sono più difficili da apprendere: servono più dati per scegliere fra le più numerose ipotesi coerenti con i dati stessi. Occorre trovare dei compromessi e/o ricorrere a metodi di apprendimento della rappresentazione ottimale. Successivamente ci si concentrerà su metodi specifici per la rappresentazione scelta (e.g., alberi di decisione, reti neurali, ecc...).
- **Ricerca:** scelti rappresentazione e bias, l'apprendimento può essere visto come un *problema di ricerca* da risolvere attraversando lo *spazio delle ipotesi* (rappresentazioni possibili del modello), dato un **search bias**, per trovare quella che meglio si adatta agli esempi. Tipicamente gli spazi di ricerca sono molto estesi, spesso infiniti, quindi risulta impossibile una ricerca sistematica e si deve ricorrere a forme di *ricerca locale*. Per la definizione dell'*algoritmo*, quindi, si richiede la definizione di uno *spazio* di ricerca, di una *funzione* di valutazione, e di un *metodo* di ricerca.
- **Dati imperfetti** nelle situazioni reali, a seguito di diverse cause quali
 - *rumore*: feature osservate che risultano inadeguate a predire la classificazione;
 - *dati mancanti*: per alcuni esempi, non sono disponibili i valori di alcune feature;
 - *errori*: i valori di alcune feature risultano errati.
- **Ordine** naturale del dominio delle feature (ad esempio nello spazio, nel tempo). L'ordine permette di distinguere problemi di **interpolazione**, in cui si devono fare predizioni per casi *ricompresi* nelle regioni dei dati osservati (training), da quelli di **estrapolazione**, in cui le predizioni riguardano casi distanti dagli esempi di training. I modelli per tali problemi risultano in genere meno accurati (si dice che è “*più facile predire il passato che il futuro*”) in quanto determinare il valore incognito di una feature in base a valori di input già osservati subito prima o dopo, risulta molto più facile che predire valori futuri. Per questo, ipotesi con parametri ottimizzati ai fini dell'*interpolazione* possono rivelarsi pessime su successivi problemi di *estrapolazione*.
- **Dimensionalità**: essendo gli esempi descritti in termini di feature (una *dimensione* per ciascuna) il numero di esempi possibili cresce esponenzialmente aumentando le dimensioni (*curse of dimensionality*). Ad esempio, un frame in un video 4K è costituito da 8M pixel (dimensione). Essendo improbabile trovare esempi molto vicini a quelli già osservati, si determina la *sparsità* dello spazio degli esempi. Al crescere della dimensionalità, le singole dimensioni diventano poco importanti: in problemi dell'ambito medico, se gli esempi sono costituiti da immagini e altri attributi (dimensioni) la modifica di un singolo pixel non dovrebbe influenzare la diagnosi.
- **Acquisizione offline/online** degli esempi: nel primo caso tutti gli esempi sono disponibili prima dell'azione/decisione, nel secondo essi si rendono disponibili nel tempo, nel corso del processo di learning. La rappresentazione degli esempi (conoscenza pregressa) potrebbe essere determinata prima di averli visti tutti (cosa non possibile, in genere) per cui per nuovi esempi si rende necessario un aggiornamento della rappresentazione. Nell'*active learning*, il sistema ragiona e decide sugli esempi *utili* da acquisire per svolgere il suo compito.

2 Fondamenti

Problema Supervisionato

Una **feature** (in italiano, *caratteristica* o *attributo*) è una funzione F definita sugli *esempi*, ossia sullo spazio della loro rappresentazione:

- $F(e)$ denota il *valore* di F per l'esempio e (e.g. $\text{lunghezza}(e) = 123$ parole);
- il *dominio* della feature è l'insieme dei valori che essa può assumere, ossia il codominio della funzione F (e.g. $\text{dom}(\text{lunghezza}) = \mathbb{N}$, o anche \mathbb{R}).

Il **problema** di apprendimento **supervisionato** si può descrivere come segue:

- *dati*:
 - le **feature di input** X_1, \dots, X_m ;
 - le **feature-obiettivo** (o **target**) Y_1, \dots, Y_k (nel seguito solo una: $k = 1$);
 - un (multi)insieme di **esempi** $e = (x_e, y_e)$ con
 $x_e = (X_1(e), \dots, X_m(e))$ e $y_e = (Y_1(e), \dots, Y_k(e))$;
- *determinare* un **predittore**, ossia una funzione atta a predire i valori delle Y_j , dati i valori delle X_i , per esempi non osservati, ossia *nuovi* rispetto a quelli dati.

Come accennato in precedenza, vi sono diverse varianti dei problemi di apprendimento supervisionato:

- p. di **regressione**: predire un valore target continuo, sottoinsieme dei numeri reali;
- p. di **classificazione**: predire un valore target ricompreso in insieme finito prefissato, ad esempio binario / booleano $\{\text{false}, \text{true}\}$, o a valori multipli, come le misure $\{XS, S, M, L, XL, \dots\}$, anche non ordinati;
- p. **relazionale**: predire relazioni, come ad esempio chi sia la madre di una data persona; in tal caso i nuovi esempi per la valutazione saranno tratti da un'altra popolazione rispetto a quelli usati nell'addestramento;
- p. **strutturale**: predire una struttura, e.g. la forma di una molecola.

Esempi di applicazioni dei modelli supervisionati:

- predire *attività* (e.g., camminare, correre, sedersi, guidare, dormire) dalla frequenza cardiaca e dal movimento così come rilevati da uno *smart-watch*;
- predire la *probabilità* di inondazione di un sito nella prossima decade in base dati topografici, climatici, geologici, sull'uso del suolo (a fini assicurativi o della concessione di permessi edilizi);
- predire *parole scritte a mano* su smartphone / tablet in base alla grafia (ordine del tratto);
- predire *testo tradotto* in Indonesiano corrispondente a uno in Swahili.

Esempio — Problema di *classificazione*: decisione sull'eventuale lettura o salto di un *post* in un dato *thread*:

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

<i>Es</i>	<i>Author</i>	<i>Thread</i>	<i>Length</i>	<i>Where_read</i>	<i>User_action</i>
<i>e₁</i>	known	new	long	home	skips
<i>e₂</i>	unknown	new	short	work	reads
<i>e₃</i>	unknown	followup	long	work	skips
<i>e₄</i>	known	followup	long	home	skips
<i>e₅</i>	known	new	short	home	reads
<i>e₆</i>	known	followup	long	work	skips
<i>e₇</i>	unknown	followup	short	work	skips
<i>e₈</i>	unknown	new	short	work	reads
<i>e₉</i>	known	followup	long	home	skips
<i>e₁₀</i>	known	new	long	work	skips
<i>e₁₁</i>	unknown	followup	short	home	skips
<i>e₁₂</i>	known	new	long	work	skips
<i>e₁₃</i>	known	followup	short	home	reads
<i>e₁₄</i>	known	new	short	work	reads
<i>e₁₅</i>	known	new	short	home	reads
<i>e₁₆</i>	known	followup	short	work	reads
<i>e₁₇</i>	known	new	short	home	reads
<i>e₁₈</i>	unknown	new	short	work	reads
<i>e₁₉</i>	unknown	new	long	work	?
<i>e₂₀</i>	unknown	followup	short	home	?

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

es.	X	Y
<i>e₁</i>	0.7	1.7
<i>e₂</i>	1.1	2.4
<i>e₃</i>	1.3	2.5
<i>e₄</i>	1.9	1.7
<i>e₅</i>	2.6	2.1
<i>e₆</i>	3.1	2.3
<i>e₇</i>	3.9	7.0
<i>e₈</i>	2.9	?
<i>e₉</i>	5.0	?

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

2.1 Valutare le Predizioni

Dato il *predittore* \hat{Y} per la feature target Y , $\hat{Y}(e)$ denoterà **stima puntuale** del valore di Y per l'esempio $e \in Es$, ossia la predizione del vero valore $Y(e)$ detto *ground truth*; il predittore \hat{Y} , appreso sulla base degli esempi di training, potrà essere applicato in seguito a qualsiasi altro esempio.

Si dice **loss** (o *errore*) di $\hat{Y}(e)$ una misura della *distanza* della stima $\hat{Y}(e)$ dal vero valore $Y(e)$. Essa viene definita come funzione $loss(p, a)$ su \mathbb{R}_+ : errore della predizione di p quando il valore effettivo (*actual*) è a . Tipicamente se ne considera la media sull'insieme Es di esempi, o **mean loss**:

$$\frac{1}{|Es|} \sum_{e \in Es} loss(\hat{Y}(e), Y(e))$$

o, in alternativa, la semplice sommatoria (senza $1/|Es|$) che ha lo stesso punto di minimo. Si preferisce la mean loss perché può essere interpretata senza conoscere il numero di esempi $|Es|$.

2.1.1 Feature Target a Valori Reali

In caso di feature con valori *totalmente ordinati*, le differenze sono significative (anche nei casi di valori interi). Ad esempio, altezze in cm, voti, numero bambini, taglie (mappate su numeri).

Le funzioni di *loss* più utilizzate in problemi di *regressione*, ossia con $a, p \in \mathbb{R}$, sono elencate di seguito:

- **errore 0-1 o loss L_0** :

$$loss(p, a) = \begin{cases} 1 & p \neq a \\ 0 & p = a \end{cases}$$

anche con la funzione $\mathbf{1}(p \neq a)$ che vale 1 se l'argomento è vero e 0 altrimenti (cfr. δ di Kronecker). La loss media su un dataset sarà il numero medio di predizioni sbagliate, che non tiene conto di *quanto* siano errate. L'**accuratezza** ne costituisce il complemento a 1, ossia il numero medio di predizioni corrette e viene massimizzata minimizzando la loss. Il test d'uguaglianza non è affidabile nel confronto fra numeri reali quindi non è utile calcolare la loss media, mentre potrebbe esserlo il calcolo di moda o mediana.

- **errore assoluto o loss L_1** :

$$loss(p, a) = |p - a|$$

È sempre non negativa e per un dataset risulta *nulla* se tutte le predizioni sono corrette; a differenza della precedente, tiene conto della *precisione* delle predizioni.

- **errore quadratico o loss L_2** :

$$loss(p, a) = (p - a)^2$$

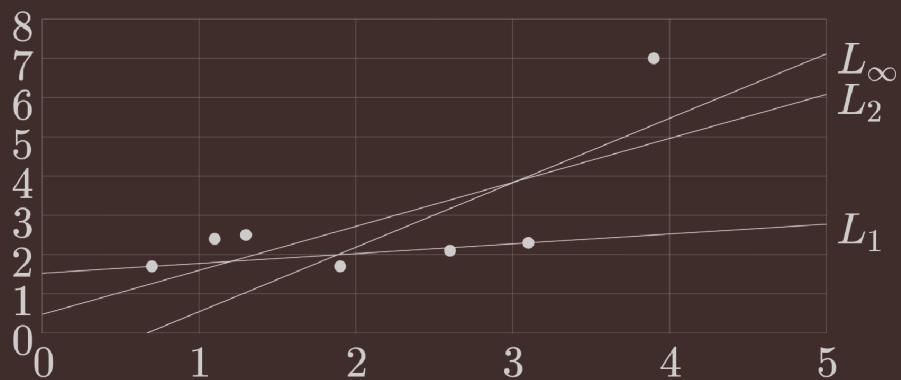
Si tratta di un errore amplificato, ad esempio un errore di 2 vale quanto 4 errori di 1 e un errore di 10 vale quanto 100 errori di 1; ai fini della minimizzazione, si considera la *loss media quadratica* su Es o, equivalentemente, la **radice dell'errore quadratico medio** (*root-mean-square error*, RMSE) che ha gli *stessi* punti di minimo delle sommatorie sul dataset; spesso si antepone il fattore $\frac{1}{2}$ per semplificare le derivate.

- **errore nel caso pessimo o loss L_∞ su Es** :

$$loss(Es) = \max_{e \in Es} |\hat{Y}(e) - Y(e)|$$

Rappresenta una stima pessimistica basata sull'errore massimo del predittore quindi non corrisponde a sommatorie o medie.

Esempio — Problema di regressione precedente



Rette utili a predire Y minimizzando l'errore L_1 , L_2 e L_∞

- ogni retta passante per due punti minimizza l'errore L_0 : le triple di esempi sono tutte *non co-lineari*;
- per il punto con $X = 1.1$ e $Y = 2.4$ effettivo, si hanno predizioni *simili* dalle rette L_1 e L_2 (rispettivamente 1.805 e 1.709) mentre L_∞ predice il valore 0.7;
- *interpolando* in [1, 3] si fanno predizioni distanti al più di 1.5;
- *estrapolando, divergono*: ad esempio per $X = 5$, L_1 e L_∞ forniscono predizioni molto differenti.

Un **outlier** è un esempio che non segue il *pattern* degli altri esempi. La differenza tra le diverse rette più evidente nel trattamento di outlier come, ad esempio, (3.9, 7):

- la predizione con L_∞ dipende solo da (1.1, 2.4), (3.1, 2.3) e (3.9, 7), con lo stesso errore di caso pessimo mentre gli altri punti sono ininfluenti se non sono più lontani dalla retta;
- la 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, rispettivamente; ad esempio, si ha stessa predizione considerando il punto (3.9, 107) anziché (3.9, 7);
- la predizione con L_2 è sensibile a tutti i dati: cambiando la Y di qualche esempio, cambia la retta e i cambiamenti negli outlier hanno maggiori effetti sulla retta rispetto a quelli relativi a punti ad essa vicini.

2.1.2 Feature Target Categorical

Una feature **categorica** Y ha un dominio finito $D = \{v_1, \dots, v_k\}$. Ad esempio, in base alla rappresentazione dei dati potrebbe rappresentare l'argomento principale di un articolo, la specie di un animale, il paese d'origine di una persona, il numero rappresentato nell'immagine di una cifra scritta a mano.

La tipica *stima puntuale* per tale tipo di feature è la **predizione certa** di uno fra i possibili valori v_1, \dots, v_k , ovvero una tupla binaria con un solo 1 alla posizione corrispondente (*one-hot encoding*). In alternativa, è possibile una **predizione probabilistica**, tramite una funzione p (o dizionario) che associa D a numeri reali non negativi in modo tale che $\sum_{v \in D} p[v] = 1$, dove $p[v]$ è valore della predizione di p per v ; la predizione certa può essere definita equivalentemente se $\exists j : p[v_j] = 1$ e p è nulla per gli altri valori.

Esempio — Imparare le preferenze sulla lunghezza di una vacanza (1-6 giorni)

- dataset semplice (senza input):

Es	Y
e_1	1
e_2	6
e_3	6
e_4	2
e_5	1

- per una Y a valori reali, la predizione per un nuovo e potrebbe essere un generico numero reale, e.g. $\hat{Y}(e) = 3.5$;
- per Y categorica, con dominio $\{1, 2, 3, 4, 5, 6\}$:
 - predizione *certa* per e : un valore del dominio, e.g. $\hat{Y}(e) = 3$;
 - predizione *probabilistica* per e : si può rappresentare $\hat{Y}(e)$ come *dizionario* p con $p[1] = 0.25$, $p[2] = 0.2$, $p[3] = 0.1$, $p[4] = 0.1$, $p[5] = 0.1$, $p[6] = 0.25$.

Al caso categorico possono essere applicate le stesse loss del caso continuo. Per $Y(e) = a$:

$$\text{loss}(p[a], 1)$$

essendo $p[a]$ la probabilità associata al caso corretto.

Ad esempio l'errore quadratico per la predizione probabilistica sarà definito: $(1 - p[a])^2$.

Una tipica misura delle prestazioni è l'**accuratezza media**:

- predizione *certa*: 1 per predizioni corrette, nulla altrimenti;
- predizione *probabilistica*: 1 quando c'è un'unica moda in corrispondenza al valore effettivo (v con $\max p[v]$ associata). Dipendendo solo dalla moda, l'accuratezza risulta una misura approssimativa per questo tipo di predizione.

La distribuzione p può essere ottimizzata anche usando la **log-loss (categorica)** o *entropia incrociata*:

$$\text{logloss}(p, a) = -\log p[a]$$

misura sempre non negativa, nulla se $p[a] = 1$ con le altre a 0. Dato il dataset Es , la **log-loss media** del predittore \hat{Y} sarà:

$$-\frac{1}{|Es|} \sum_e \log \hat{Y}(e)[Y(e)]$$

dove $\hat{Y}(e)[Y(e)]$ è il valore della predizione $p = \hat{Y}(e)$ per l'effettivo $a = Y(e)$.

Esempio — Predizioni probabilistiche per il dataset visto in precedenza: $\{(e_1, 1), (e_2, 6), (e_3, 6), (e_4, 2), (e_5, 1)\}$

Se la predizione per gli esempi e_1, e_2, e_3, e_5 avesse probabilità $0.25 = p[1] = p[6]$, mentre per e_4 questa fosse di $0.2 = p[2]$, la log-loss media sarà $-\frac{1}{5}(4 \cdot \log_2 0.25 + \log_2 0.2) \approx 2.064$;

Se le predizioni avessero probabilità $p[1] = p[6] = 0.4$ e $p[2] = 0.2$ e nulle per gli altri valori (3, 4, 5), la log-loss media sarà $-\frac{1}{5}(4 \cdot \log_2 0.4 + \log_2 0.2) \approx 1.522$.

Si noti che la seconda predizione, che adotta la proporzione osservata nel dataset per ciascun valore, risulta essere migliore della prima.

Il miglior predittore è un modello per il quale si massimizza la probabilità dei dati osservati, ossia la *verosimiglianza* degli esempi E_s dato \hat{Y} :

$$\prod_{e \in E_s} \hat{Y}(e)[Y(e)]$$

assumendo che gli esempi siano fra loro indipendenti. Passando al logaritmo, si ottiene la cosiddetta **log-likelihood**:

$$\sum_{e \in E_s} \log \hat{Y}(e)[Y(e)]$$

Quindi la *log-loss media* si ottiene cambiando il segno alla log-likelihood e dividendo per il numero di esempi. Essa sarà minima quando la log-likelihood (o la likelihood) è massima.

La log-loss media risulta appropriata per predizioni nel ragionamento sotto incertezza; ad esempio, sotto loss quadratica, i valori 10^{-7} e 10^{-6} sono molto vicini: una predizione di 10^{-7} avrà un errore *simile* a quello di 10^{-6} . Invece, in termini di probabilità, rappresenterebbero casi molto *diversi*. Tale differenza può essere apprezzata adottando la log-loss ma non con le loss basate sulle differenze fra valori predetto ed effettivo.

Essendo $\log 0$ indefinito, la log-loss per la predizione di 0 è (conventionalmente) trattata come fosse infinita. Dovrebbe perciò essere scoraggiata qualsiasi predizione di probabilità nulla per un valore che invece possa essere osservato per qualche esempio futuro (cfr. §4.1). La log-loss è correlata con l'*entropia*: numero di simboli (*bit*) per codificare i dati, con un codice (binario) basato su \hat{Y} come distribuzione di probabilità (passando a logaritmi in base e si parla di *nat*).

2.1.3 Feature Target Binarie

Si consideri il caso in cui la feature target Y abbia il dominio $\{0, 1\}$ (o anche $\{\text{false}, \text{true}\}$); tale valore potrebbe indicare, ad esempio, uno degli argomenti di un articolo di giornale, usando una variabile booleana Y_k per ciascun argomento possibile.

In tal caso la predizione può essere data da un valore reale: $p = \hat{Y}(e) \in [0, 1]$, il che è equivalente alla predizione categorica¹ di p per il valore 1 (ovvero di $1 - p$ per 0).

Si definisce la **log-loss binaria**, o *cross-entropy loss binaria*, come mostrato di seguito:

$$\text{logloss}(p, a) = -[a \log p + (1 - a) \log(1 - p)]$$

dove p è la predizione e a il valore effettivo, ricavabile dalla verosimiglianza². Per convenzione $\text{logloss}(1, 1) = \text{logloss}(0, 0) = 0$, anche se $\log 0$ non è definito (infinito al limite). La log-loss sarà pari a $\log p$ se $a = 1$, altrimenti sarà $\log(1 - p)$, come la log-loss categorica per la quale la predizione per 1 è p e per 0 è $1 - p$. Il termine *log-loss* le ricopre entrambe.

Teoria dell'Informazione — Richiami

Cfr. specchietto in [PM23] e approfondimenti in [Fla12, McK05].

Codifica Binaria: Il **bit** è la quantità d'informazione utile codificare 2 simboli distinti. Con l bit (lunghezza) si formano 2^l codici binari che distinguono altrettanti simboli. Per distinguere n simboli (problema inverso), serve una lunghezza dei relativi codici pari a $\log_2 n$ bit. Tale lunghezza n serve nel caso si possa assumere che i simboli siano *equiprobabili*: $\forall x P(x) = \frac{1}{n}$ (costante).

In generale, nota la *distribuzione* P dei simboli, per codificare x , la lunghezza ottimale sarà:

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

I simboli più probabili (frequenti) richiedono codici più corti.

L'entropia (o *contenuto informativo*) di $P(x)$ misura il numero *medio* di bit utile a trasmettere una *sequenza codificata* in un alfabeto binario: ognuno richiede in media

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

Data l'evidenza e , si può definire anche la nozione di *entropia condizionata* di $P(x|e)$:

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

L'*informazione attesa* di un *test a 2 vie*, ossia di una condizione α (e.g. della forma $F(\cdot) = v$) è l'entropia condizionata calcolata mediando sui due esiti possibili, α e $\neg\alpha$:

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

L'**information gain** (IG) di un test è il guadagno dovuto al test e calcolato come differenza tra entropia iniziale e informazione attesa dopo il test.

2.2 Baseline: Stime Puntuale senza Feature di Input

Si consideri il *caso* in cui la predizione sia basata solo sui valori della feature-obiettivo Y osservata per i diversi esempi del dataset, senza usare le feature di input X_i . Si avrà un modello *costante* (ossia una **baseline** naive) per il quale si predice *uno stesso valore* $v = \hat{Y}(e)$ per tutti gli esempi e . Il valore ottimale da predire si sceglie in base al criterio da ottimizzare (la loss) e potrà essere calcolato e ottimizzato *analiticamente* sulla base degli n esempi di training.

Nel caso di Y a *valori reali*, il modello fornisce la stima puntuale $p = \hat{Y}(e) \quad \forall e$:

Criterio	Errore per p	Predizione ottimale
loss 0-1 media	$\frac{1}{n} \sum_e \mathbf{1}(Y(e) \neq p)$	moda
loss assoluta media	$\frac{1}{n} \sum_e Y(e) - p $	mediana
loss quadratica media	$\frac{1}{n} \sum_e (Y(e) - p)^2$	media
loss caso pessimo	$\max_e Y(e) - p $	$(\min + \max)/2$

Per una feature target Y *binaria*, i dati possono essere riassunti da due semplici statistiche $n_1 = \sum_e Y(e)$ e $n_0 = n - n_1$. La media o **frequenza empirica**, ossia la proporzione di 1 nel training set è $\frac{n_1}{n}$. Essa costituisce anche una *stima di massima verosimiglianza* quindi anche la predizione con la minima log-loss. Gli errori 0-1 ed assoluto sono minimizzati considerando il valore binario più frequente.

Criterio	Errore per p	Predizione ottimale
log-loss media	$-\left[\frac{n_1}{n} \log p + \frac{n_0}{n} \log(1-p)\right]$	$\frac{n_1}{n}$

Nel caso di una Y categorica, con un dominio $\{v_1, \dots, v_k\}$, il dataset può essere sintetizzato dalle frequenze n_1, \dots, n_k . La log-loss sarà minimizzata dalla predizione della frequenza empirica sul training set. L'errore 0-1 viene minimizzato dalla predizione della moda, che porta anche a massimizzare l'accuratezza.

Criterio	Errore per p	Predizione ottimale
log-loss media	$-\sum_i \frac{n_i}{n} \log p_i$	$p[v_i] = \frac{n_i}{n}$

Esempio — Dati preferenze vacanze: considerando solo i valori di Y

1. come feature a valori reali:
 - la predizione di 1 (o di 6) minimizza l'errore 0-1 e massimizza l'accuratezza sui dati di training;
 - la predizione di 2 minimizza la loss assoluta per cui la loss totale è 10 e loss assoluta media è 2;
 - 3.2 è la predizione che minimizza l'errore quadratico;
 - 3.5 è la predizione che minimizza l'errore del caso pessimo;
2. come feature categorica:
 - predizione che minimizza la log-loss: $(0.4, 0.2, 0, 0, 0, 0.4)$ distribuzione empirica dei dati di training.

Quindi la predizione preferibile dipende dalla rappresentazione di Y e dalla misura da ottimizzare. Per esempi non ancora osservati la frequenza empirica può risultare problematica se si massimizza la likelihood o minimizza la log-loss. Ad esempio se $n_i = 0$ allora il valore v_i sarà da considerarsi mai osservato. Un nuovo esempio di test con un valore v_i comporterebbe una likelihood nulla (valore minimo) e una log-loss infinita (non definita), costituendo un caso di *sovradattamento* (trattato più avanti).

Come *baseline* di partenza si possono considerare predizioni *naive*, determinate *prima* di osservare i dati (sfruttando conoscenza pregressa, il parere degli esperti del dominio).

Per una variabile binaria gli errori-base da battere nei vari casi sono i seguenti:

Criterio	Pred. Naive	Loss media
0-1 loss media	1	≤ 1
loss assoluta media	0.5	0.5
loss quadratica media	0.5	0.25
errore caso pessimo	0.5	0.5
log-loss media (base 2)	0.5	1
log-loss media (base e)	0.5	0.693..

Tali predizioni sono utili a testare rapidamente un metodo (sulla base del training set, ma ignorando le X_i). Ad esempio un predittore per Y binaria con loss quadratica media > 0.25 risulterebbe non buono. Occorrerebbe battere la loss media per avere la migliore predizione dato il criterio usato, un compito che spesso risulta difficile.

2.3 Tipi di Errore

Gli errori non sono tutti uguali, alcuni tipi possono essere peggiori di altri in base alle loro conseguenze.

Esempio — Medicina

Non diagnosticare una malattia a un paziente, che sarà indotto a non curarsi, può risultare più grave rispetto a predire una malattia che non ha, per cui sarebbero consigliati ulteriori accertamenti.

Per un esperimento si può compilare una **matrice di confusione**. Nel caso più semplice si supponga che Y sia booleana. La valutazione della predizione sarà 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:

- *falsi positivi* o errore di *I tipo*: predizione positive sbagliate (*true* invece dell'effettivo valore *false*);
- *falsi negativi* o errore di *II tipo*: predizione negative sbagliate (*false* invece dell'effettivo *true*).

Le tipiche misure definite per un **modello predittivo binario** sono le seguenti:

- **precisione** (*precision*): $\frac{tp}{tp + fp}$
proporzione di veri positivi (*tp*) rispetto ai casi predetti come positivi (*pp*);
- **richiamo** (*recall* o *true-positive rate*, TPR): $\frac{tp}{tp + fn}$
proporzione di veri positivi (*tp*) rispetto a tutti i positivi effettivi (*ap*);
- analogamente *false-positive rate* (FPR): $\frac{fp}{fp + tn}$
proporzione di falsi positivi (*fp*) su tutti i negativi (*an*).

Nella scelta del modello predittivo occorrerebbe massimizzare precisione e richiamo e minimizzare il FPR, ma nella pratica tali obiettivi risultano *incompatibili*: si può massimizzare la precisione e minimizzare il FPR facendo predizioni positive solo se si è sicuri, ma ciò abbassa il richiamo; per massimizzare il richiamo si possono ammettere previsioni più azzardate ma questo tende ad abbassare la precisione e a incrementare il FPR.

Si supponga a titolo d'esempio che i falsi positivi rappresentino casi c volte peggiori dei falsi negativi:

$c < 1$	$c = 1$	$c > 1$
FP preferibili ai FN	parità	FP peggiori dei FN

Si tratta di costi *additivi*, quindi il **costo** di una predizione sarebbe proporzionale a $c \cdot fp + fn$. Si dovrebbe scegliere il predittore con il costo più basso.

Nel confronto di modelli, si dice che un predittore A **domina** B se A ha più TPR e meno FPR di B . In tal caso A avrà un costo inferiore (sarà migliore) di B per tutte le funzioni che dipendono solo da numero di FP e FN, assumendo un costi additivi non negativi.

Indipendentemente dai costi, si possono considerare lo **spazio ROC** (*receiver operating characteristic*), ossia il grafico del FPR contro il TPR e lo **spazio P-R**, ossia il grafico della precisione contro il richiamo. Ogni modello produce un punto in ciascuno 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

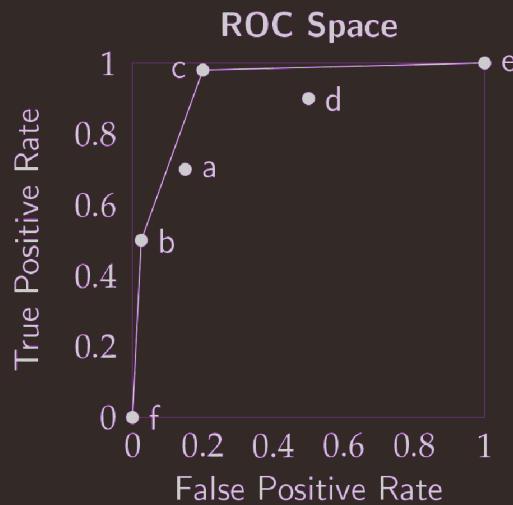
b	ap	an
pp	50	25
pn	50	975

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**: TPR, richiamo 0.7, FPR 0.15, precisione $70/220 \approx 0.318$;
- **c**: TPR, richiamo 0.98, FPR 0.2, precisione $98/298 \approx 0.329$

per cui **c** risulta 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, allora **a** sarebbe preferibile rispetto a **c** (si vede nello spazio ROC, ma non in quello P-R). Risulta peggiore un modello più in basso e più a destra di altri, ad esempio **d** è peggiore di **c**: non sarebbe logico preferire **d** a **c**.

Per alcuni modelli, è possibile individuare un *iperparametro* che, variato, porta a diversi valori di TPR e FPR. Un modello che restituisce un valore reale potrebbe predire **true** o **false** in base al superamento di una *soglia* (parametro). Tale modello può essere rappresentato con una curva nello spazio ROC al variare di tale parametro e si potrà dire che domina un altro modello se la sua curva è al di sopra e più a sinistra. Se la situazione è più complessa si può ricorrere a funzioni di costo che dipendono dai due tipi di errore oppure si può adottare l'**area sotto la curva** ROC (AUROC o AUC) una misura numerica per confrontare modelli nello spazio dei parametri (sotto una particolare funzione di costo). Risulterà migliore il modello con l'AUC maggiore.

3 Modelli-Base

I **modelli di predizione** da apprendere sono funzioni (nel seguito saranno chiamati anche *classificatori* e *regressori*). Schematicamente, l'obiettivo di un *algoritmo supervisionato* può essere sintetizzato come segue:

- **dato** un insieme di esempi descritti in termini delle feature di input e target

- **restituire** la rappresentazione compatta di una funzione utile a predire il valore della feature target di nuovi esempi, per i quali siano disponibili solo valori delle feature di input.

Le differenze fra i diversi algoritmi dipenderanno dalla scelta fra le diverse possibili rappresentazioni delle funzioni. Nel seguito, si prenderanno in considerazione *modelli base*, essenzialmente *funzioni costanti a pezzi* o *lineari*, che torneranno utili a costruirne di più complessi.

3.1 Alberi di Decisione

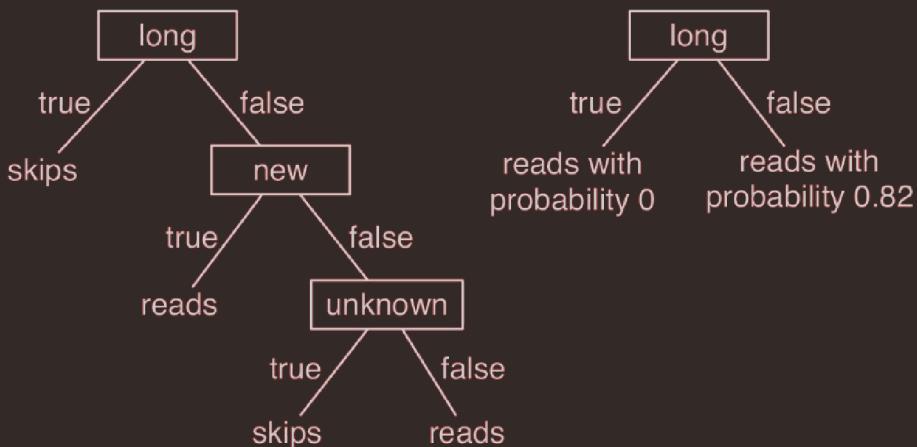
Un **albero di decisione** è albero binario costituito da:

- nodi *interni* (non-foglia) etichettati con *condizioni*, ossia test booleani³ basati sui valori delle feature negli esempi e connessi con due *figli*, radici di sotto-alberi, attraverso archi tipicamente etichettati con **true** e **false**;
- nodi-*foglia* etichettati con una stima puntuale per la feature obiettivo: la classe in un *albero di classificazione* oppure un valore reale in un *albero di regressione*.

Tale struttura rappresenta una *funzione* definita *per parti* ovvero un *programma* fatto di strutture *condizionali* annidate.

Per la *predizione*, dato un esempio con i valori delle variabili di input: partendo dalla radice, ogni condizione incontrata viene valutata per tale esempio seguendo l'arco corrispondente al risultato di ogni test; raggiunta una foglia, si restituisce la stima puntuale associata.

Esempio — Due alberi di decisione per il dataset precedente sui post



Funzione di classificazione per l'albero a sinistra:

```

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

L'albero a destra ammette predizioni probabilistiche: probabilità di *UserAction = reads*

```

define UserAction(e):
    if long(e)
        return 0
    else
        return 0.82
  
```

Le due questioni principali nell'apprendimento di alberi di decisione sono le seguenti: dati gli esempi di training

- QUALE albero generare? dovrebbe essere possibile rappresentare *qualsiasi funzione* con feature X_i discrete; il *bias* è insito nel criterio di preferenza tra i diversi alberi: andrebbero favoriti alberi *più piccoli* (coerenti con i dati), ossia con la minima profondità o con meno nodi.
- COME generarlo? Si può effettuare una *ricerca* in uno spazio di alberi a partire dal più piccolo che si adatti ai dati, ma tale spazio è enorme per cui una ricerca esaustiva non è proponibile; ci si affida, quindi, a un metodo di ricerca *greedy*, puntando a minimizzare l'errore.

Ricerca di Buoni Alberi di Decisione

Come detto si può considerare un albero di decisione come *programma* che sappia determinare la predizione per qualunque esempio e arbitrario: ciascuna *foglia* conterrà una predizione del valore di Y (senza condizioni); i nodi *interni* conterranno test di condizioni seguiti da diramazioni come in `if $c(e)$ then $t_1(e)$ else $t_0(e)$` , dove c è la condizione booleana da testare e t_1 e t_0 sono i suoi sotto-alberi, ovvero sotto-programmi, t_1 sul ramo `true` e t_0 sul ramo `false`.

L'algoritmo segue (top-down) la decomposizione *ricorsiva* della struttura dell'albero:

```
procedure DT_Learner( $C_s, Y, E_s, \gamma$ )
  Input
     $C_s$ : insieme delle possibili condizioni
     $Y$ : feature-obiettivo
     $E_s$ : insieme di esempi di training
     $\gamma$ : miglioramento minimo per l'espansione ( $\gamma \geq 0$ )
  Output
    funzione per predire il valore di  $Y$  per un esempio

   $c \leftarrow select\_split(E_s, C_s, \gamma)$ 
  if  $c = None$  // criterio di stop raggiunto
     $v \leftarrow leaf\_value(E_s)$ 
    define  $T(e) = v$ 
    return  $T$ 
  else
     $true\_examples \leftarrow \{e \in E_s : c(e)\}$ 
     $t_1 \leftarrow DT\_learner(C_s \setminus \{c\}, Y, true\_examples, \gamma)$ 
     $false\_examples \leftarrow \{e \in E_s : \neg c(e)\}$ 
     $t_0 \leftarrow DT\_learner(C_s \setminus \{c\}, Y, false\_examples, \gamma)$ 
    define  $T(e) = \text{if } c(e) \text{ then } t_1(e) \text{ else } t_0(e)$ 
    return  $T$ 

  procedure select_split( $E_s, C_s, \gamma$ )
     $best\_val \leftarrow sum\_loss(E_s) - \gamma$ 
     $best\_split \leftarrow None$ 
    for-each  $c \in C_s$  do
       $val \leftarrow sum\_loss(\{e \in E_s | c(e)\}) + sum\_loss(\{e \in E_s | \neg c(e)\})$ 
      if  $val < best\_val$  then
         $best\_val \leftarrow val$ 
         $best\_split \leftarrow c$ 
    return  $best\_split$  //  $best\_split = None \rightarrow$  criterio di stop raggiunto
```

Nell'algoritmo si effettua una scelta miope della condizione c ottimale — **greedy optimal split** — basata sulla somma delle loss prima e dopo il partizionamento e una soglia γ ; l'aggiunta di un livello/partizione aumenta le dimensioni dell'albero di un'unità mentre γ penalizza tale incremento: se positiva, essa è utile a evitare che si verifichi $val < best_val$ per meri errori di arrotondamento.

Il *criterio di stop* viene determinato dall'esaurimento delle condizioni ovvero da esempi che risultano (tutti) d'una stessa classe, il che porta ad avere una loss nulla. L'*output* (nelle foglie) rappresenta una stima *puntuale* (dipende solo dalle condizioni incontrate nel percorso). Come stima puntuale si sceglie, in genere, il valore più *probabile*, la moda, oppure la mediana o la media; come visto in precedenza, in alternativa si può optare per la restituzione di una distribuzione di probabilità delle classi per gli esempi finiti in quella foglia, che corrisponderà a una regione ristretta dello spazio degli esempi.

Esempio — Usando l'algoritmo sui dati visti in precedenza:

- $\text{DT_Learner}(\{\text{known}, \text{new}, \text{long}, \text{home}\}, \text{User_action}, \{e_1, e_2, \dots, e_{18}\}, 0)$
si supponga che il criterio di stop non si avveri e si scelga il test: $\text{Length} = \text{long}$
- chiamata per t_1 :
 $\text{DT_Learner}(\{\text{known}, \text{new}, \text{home}\}, \text{User_action}, \{e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}\}, 0)$
 - tutti gli esempi concordano sulla classe *User_action* quindi t_1 si riduce a un nodo-foglia *skips*;
- chiamata per t_0 :
 $\text{DT_Learner}(\{\text{known}, \text{new}, \text{home}\}, \text{User_action}, \{e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}, 0)$
 - gli esempi (con $\text{Length} = \text{short}$) non concordano sulla classe;
 - se non scatta il criterio di stop, si deve scegliere un nuovo test, ad esempio $\text{Thread} = \text{new}$; continuando, alla fine la chiamata per t_0 (ramo *short*, i.e. $\neg \text{long}$) restituisce:

```

if new(e) then reads
else
    if unknown(e) then skips else reads

```

Il risultato finale è l'albero / programma visto in precedenza.

Usando log-loss con base 2, la media delle loss $\text{sum_losses}(Es)/|Es|$ corrisponde all'*entropia* della distribuzione empirica di *Es*. Se *val* è il numero di bit per descrivere *Es* dopo il test di *c* allora l'entropia della distribuzione creata dal partizionamento sarà $\text{val}/|Es|$. Si definisce **information gain** (IG) la differenza fra le due entropie. Questo viene spesso usato anche assieme ad altre misure da ottimizzare, ad esempio si può puntare a massimizzare l'accuratezza selezionando la condizione *c* in base alla log-loss, ma restituendo la *moda* come valore per le foglie.

Esempio — Dataset precedente: operando una scelta (miope) del test, ai fini di minimizzare la log-loss, ovvero massimizzare la likelihood, si ha

- prima del test iniziale, la predizione ottimale basata sulla *frequenza empirica*:
 - $p[0] = |\{e \in Es : \text{User_action}(e) = \text{skips}\}|/18 = \frac{9}{18} = 0.5$
 - $p[1] = |\{e \in Es : \text{User_action}(e) = \text{reads}\}|/18 = \frac{9}{18} = 0.5$
 - per cui la log-loss è $-(9 \cdot \log_2 0.5 + 9 \cdot \log_2 0.5)/18 = -(18 \cdot \log_2 0.5)/18 = 1$
- test su *Author*: $[e_1, e_4, e_5, e_6, e_9, e_{10}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}]$ sono gli esempi con *Author* = *known* mentre $[e_2, e_3, e_7, e_8, e_{11}, e_{18}]$ sono quelli con *Author* = *unknown*
 - in ciascuna partizione si ha la stessa frequenza di esempi delle 2 classi, per cui la predizione ottimale è 0.5 per entrambe con log-loss 1; non essendoci alcuna diminuzione si verifica che tale test non aiuti la predizione;
- test su *Thread*: $[e_1, e_2, e_5, e_8, e_{10}, e_{12}, e_{14}, e_{15}, e_{17}, e_{18}]$ sono gli esempi con *Thread* = *new* e $[e_3, e_4, e_6, e_7, e_9, e_{11}, e_{13}, e_{16}]$ sono quelli con *Thread* = *followup*
 - nella partizione *new*: 3 esempi di classe *skips* e 7 di classe *reads*, per cui la predizione di *reads* ha una probabilità stimata di $7/10$;
 - nella partizione *followup*: 2 esempi di classe *reads* e 6 di classe *skips*, per cui la predizione di *reads* ha una probabilità di $2/8$;
 - la log-loss *dopo* lo split sarà:

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

- test su *Length*: $[e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}]$ sono gli esempi con *Length = long* e $[e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}]$ sono quelli con *Length = short*
 - nella partizione *long*, tutti esempi con la stessa classificazione (*skips*) per cui la predizione ha probabilità 1;
 - nella partizione *short* si osserva la proporzione di 9 : 2 tra le classi;
 - la log-loss è $-(7 \cdot \log_2 1 + 9 \cdot \log_2 9/11 + 2 \cdot \log_2 2/11)/18 \approx 7.5/18 \approx 0.417$

Quindi per la radice il test su *Length* risulta il migliore.

Costruzione delle Condizioni

Se le feature di input sono booleane è facile definire le condizioni per i test. In caso di feature non booleane ci sono diverse possibilità:

- se X è categorica, con dominio $\{v_1, \dots, v_k\}$, si può associare una **variabile indicatrice** binaria X_i a ciascun valore: $X_i(e) = 1$ se $X(e) = v_i$ e $X_i(e) = 0$ altrimenti: per ogni esempio e , una sola fra $X_1(e), \dots, X_k(e)$ vale 1 (le altre saranno nulle);
- se X è **ordinale** (a valori reali o discreti) allora essa ha un dominio *totalmente ordinato* (ad esempio le taglie, il grado di istruzione); in tal caso, per ciascun valore v , si può introdurre una **variabile** booleana di **taglio/cut** che varrà 1 se $X > v$ e 0 altrimenti; da una combinazione di tagli si possono definire intervalli, e.g. il caso in cui la variabile per $X > 9$ sia vera e quella per $X > 17$ sia falsa corrisponde a $9 < X \leq 17$; valori di taglio ottimali possono essere trovati ordinando gli esempi in base al valore di X e selezionando quei valori che partizionino al meglio rispetto alla Y .
- **binning**: dato un insieme di soglie si crea una feature per ogni intervallo / *bin*: date le soglie $\alpha_1 < \alpha_2 < \dots < \alpha_k$ si ottengono $k+1$ feature booleane, una vera per X tale che $X \leq \alpha_1$, una per $\alpha_k < X$ e una per $\alpha_i < X \leq \alpha_{i+1}$ per ogni $1 \leq i < k$; si noti che il bin $\alpha_i < X \leq \alpha_{i+1}$ richiede 2 partizionamenti tramite tagli; i valori α_i possono essere predefiniti, ad esempio si possono adottare i *percentili* rilevati sui dati di training, oppure scelti in base a quelli della Y .
- per feature X categoriche, si possono definire condizioni $X \in S$ con $S \subset \text{dom}(X)$ dato l'*insieme* di valori $|S| \geq 1$; se Y è booleana per scegliere gli S , avendo ordinato $\text{dom}(X)$ in base alle proporzioni di Y veri, si trova la sezione migliore, *greedy optimal split*, della lista ordinata;
- si può pensare anche a un'estensione che preveda la possibilità di condizioni n-arie (*multiway split*) quindi per variabili con più valori nel dominio si potrà considerare un figlio per ciascun valore: si ottiene un *albero n-ario*, con struttura più complessa di quelli con test della forma **if-then-else**, più difficili da apprendere in quanto per alcuni valori delle condizioni potrebbe non essere disponibile alcun esempio di training; con euristiche greedy, come IG, una condizione su variabile con domini più grandi porta ad avere più figli e a un miglior adattamento ai dati a scapito della compattezza del modello; si osservi ad esempio che una condizione a 4 vie equivale a 3 split binari per cui si potrebbero avere 4 foglie.

Scelte di Progetto Alternative

Quando *select_split* restituisce *None* non si procede a ulteriore partizionamento. Ciò può accadere a seguito dell'esaurimento degli esempi o delle condizioni, oppure quando gli esempi hanno tutti lo stesso valore per ciascuna condizione, o hanno tutti la stessa classe. Si può avere un miglioramento con l'aggiunta di una condizione con una soglia $< \gamma$. Altri criteri di stop anticipato che eviti ulteriori partizionamenti possono essere la *minimum child size*, allorché un nodo-figlio abbia meno esempi rispetto a una data soglia, o la *maximum depth*, che segnala il raggiungimento di una profondità (altezza) massima.

È possibile che una condizione funzioni bene solo in congiunzione con altre ma un metodo greedy potrebbe non funzionare quando questo si verifica. Si tratta di un *caso difficile*: ad esempio si consideri di dover trovare un albero che approssimi una funzione di parità di k variabili booleane, vera se un numero dispari (o pari) di esse sono vere; conoscere i valori di meno di k variabili non dà alcuna informazione sul valore della funzione; si possono approssimare funzioni di parità più semplici ($k = 2$) come lo XOR o l'equivalenza, ma in generale gli alberi di decisione possono essere molto complessi. Quando dal partizionamento greedy risultano alberi troppo complicati conviene procedere alla loro *semplificazione* (ad esempio con tecniche di *pruning*).

Esempio — Si consideri un dataset con feature (binarie) di input x, y, z e target t per la funzione: $\text{if } x \text{ then } t = y \text{ else } t = z$

- ossia t è vera se x è vera e y è vera oppure se x è falsa e z è vera.

x	y	z	t
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Si possono costruire gli alberi:



L'albero a *sinistra* risulta migliore (più semplice) ma parte da x non informativa: stessa proporzione di esempi con t vera per x vera o falsa. L'algoritmo partizionerebbe su y , come nell'albero a *destra*: se y è vera allora la proporzione di t /vere risulta più grande rispetto ai casi in cui y è falsa; quando y è vera, partizionare su x predice perfettamente t quando x è vera: seguendo i percorsi verso $t = 1$ nell'albero destro si osserva che t è vera se $(x \wedge y) \vee (y \wedge \neg x \wedge z) \vee (\neg y \wedge \neg x \wedge z)$ da cui, semplificando $(x \wedge y) \vee (\neg x \wedge z)$ da cui si ottiene facilmente l'albero a sinistra.

3.2 Regressione e Classificazione Lineari

Si prenderanno in considerazione algoritmi che apprendono *modelli lineari* per risolvere problemi di regressione (target continuo) e classificazione (target discreto).

Con la **regressione lineare** si mira all'adattamento di una funzione *lineare* a un training set con feature *numeriche*. Si supponga che le feature di input siano $X_1, \dots, X_m \in \mathbb{R}$ e che la feature target sia $Y \in \mathbb{R}$.

Il modello è una **funzione lineare** delle feature di input:

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

dove $\bar{w} = \langle w_0, w_1, \dots, w_m \rangle$ vettore di **pesi** che costituiscono i *parametri* da determinare. Si ottiene una forma più compatta assumendo la feature aggiuntiva costante $X_0 = 1$.

Per definire il problema di ottimizzazione, dato l'insieme di esempi Es , si considera l'errore quadratico medio su Es per Y :

$$\begin{aligned} \text{error}(Es, \bar{w}) &= \frac{1}{|Es|} \sum_{e \in Es} \left(Y(e) - \hat{Y}_{\bar{w}}(e) \right)^2 \\ &= \frac{1}{|Es|} \sum_{e \in Es} \left(Y(e) - \sum_{i=0}^m w_i \cdot X_i(e) \right)^2 \end{aligned}$$

Questa funzione ha un unico minimo. Considerate le derivate parziali rispetto ai pesi

$$\frac{\partial}{\partial w_i} \text{error}(Es, \bar{w}) = \frac{1}{|Es|} \sum_{e \in Es} 2\delta(e) \cdot X_i(e)$$

dove $\delta(e) = \hat{Y}_{\bar{w}}(e) - Y(e)$ è una funzione lineare sui pesi. Il minimo può essere calcolato *analiticamente* risolvendo le equazioni lineari ottenute annullando tali derivate.

Funzioni Lineari Comprese

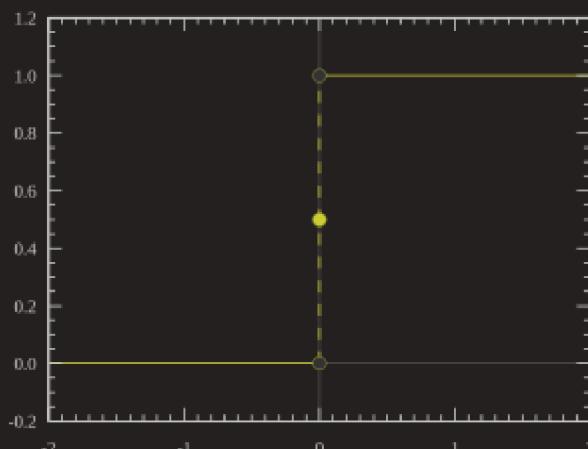
Per la *classificazione binaria* si ha una feature target Y con dominio $\{0, 1\}$. Come modello non basta una funzione lineare: le predizioni non possono superare 1 o essere minori di 0. Si considera allora una **funzione lineare compressa** (*squashed*):

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

dove ϕ è detta anche **funzione di attivazione** da \mathbb{R} (retta reale) a $R \subset \mathbb{R}$, tipicamente si considera $R = [0, 1]$. Un predittore basato su tale funzione si dice **classificatore lineare**.

Lo **step** o *gradino di Heaviside* è una funzione di attivazione semplice definita come segue:

$$\text{step}_0(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

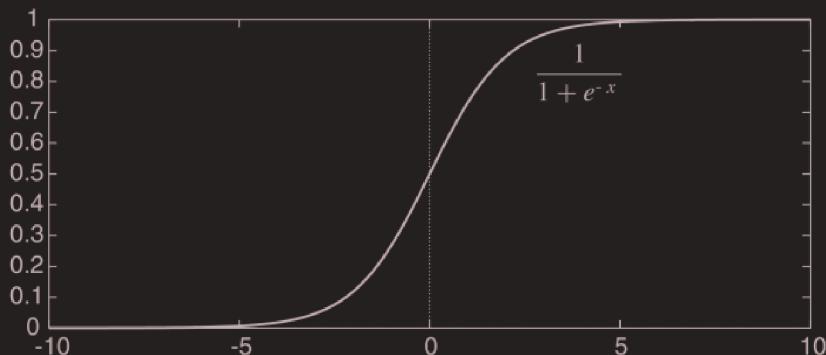


Essa viene impiegata in unità alla base dei **perceptroni** [Ros58] (cfr. reti neurali, cap. 8). Tuttavia risulta difficile adattare la discesa di gradiente (GD) a funzioni non ovunque differenziabili: con una funzione quasi ovunque differenziabile come **step**, l'ampiezza dello scalino dovrebbe tendere a zero per garantire la convergenza.

Si considerano allora in alternativa funzioni come la sigmoide (un tipo di funzione *logistica*):

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

il cui grafico è rappresentato come segue:



Essa comprime le predizioni nell'intervallo $[0, 1]$, appropriato ai fini della classificazione, e risulta differenziabile, con derivata:

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

Nella **regressione logistica** il modello di *classificazione* è costituito da una funzione lineare compressa dalla sigmoide

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

Per trovare i *pesi* \bar{w} si minimizza la *log-loss* (ovvero la *log-likelihood negativa* di cui si è trattato in precedenza) media:

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

con derivate parziali:

$$\frac{\partial}{\partial w_i} LL(Es, \bar{w}) = \frac{1}{|Es|} \sum_{e \in Es} \delta(e) \cdot X_i(e)$$

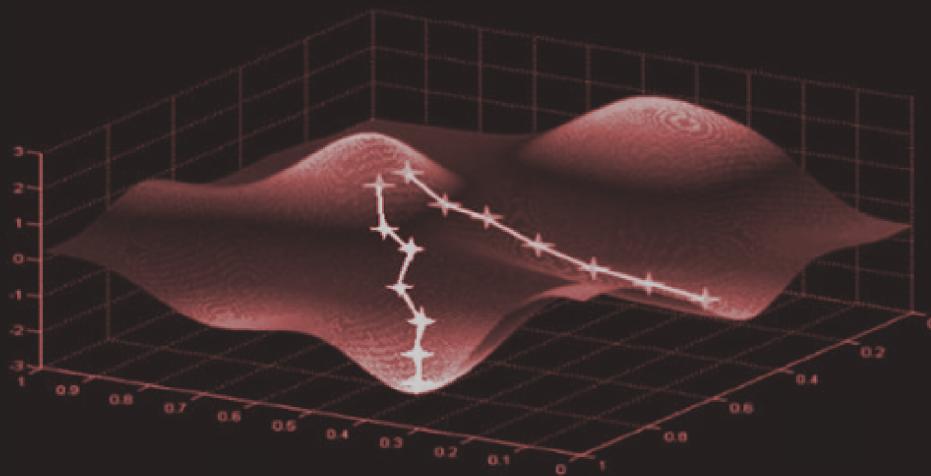
dove $\delta(e) = \hat{Y}_{\bar{w}}(e) - Y(e)$.

Tale funzione risulta difficile da minimizzare analiticamente, essendo $\hat{Y}_{\bar{w}}$ non lineare nei parametri per cui si ricorre a una soluzione algoritmica.

Discesa di Gradiente Stocastica

La **discesa di gradiente** è un metodo *iterativo* per la ricerca (locale vista nel cap. 4) dei minimi di funzioni d'errore o *loss*.

Un esempio di utilizzo dell'algoritmo è riportato in figura:



Dopo aver inizializzato \bar{w} in maniera casuale, a ogni iterata, ogni w_i verrà decrementato in proporzione alla sua derivata parziale

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

dove η è detta *passo* o **learning rate** (parametro dell'algoritmo). Le derivate misurano l'influenza sull'errore di piccole variazioni dei pesi.

Per un modello lineare, l'errore quadratico risulta una funzione *convessa* con *unico* minimo locale, quindi anche globale. Con un passo sufficientemente piccolo, l'algoritmo converge al minimo locale. Le derivate parziali di $\text{error}(Es, \bar{w})$ o log-loss media si possono scrivere come segue:

$$\frac{\partial}{\partial w_i} \text{error}(Es, \bar{w}) = \frac{1}{|Es|} \sum_{e \in Es} \delta(e) \cdot X_i(e)$$

con $\delta(e) = \hat{Y}_{\bar{w}}(e) - Y(e)$ e costante 2 assorbita da η .

Normalmente l'aggiornamento dei pesi avviene *dopo* aver iterato su *tutti* gli esempi; in alternativa, si può ricorrere alla **discesa di gradiente stocastica** (**SGD**) nella quale si prevede un aggiornamento dopo aver visto un *campione* di b esempi, detto **batch** o **minibatch**; in tal caso un passaggio completo sul dataset si chiama **epoca**, costituita da $\lceil |Es|/b \rceil$ batch.

L'algoritmo di apprendimento del modello è descritto di seguito:

```

procedure Linear_learner( $X_s, Y, Es, \eta, b$ )
  Input
     $X_s$ : insieme di feature input,  $X_s = \{X_1, \dots, X_m\}$ ,  $X_0 = 1$ 
     $Y$ : feature target
     $Es$ : insieme di esempi di training
     $\eta$ : learning rate
     $b$ : dimensione del batch
  Output
    funzione di predizione su esempi
  Local
     $w_0, \dots, w_m$ : reali
     $d_0, \dots, d_m$ : reali
  inizializzare casualmente  $w_0, \dots, w_m$ 
  define  $pred(e) = \phi(\sum_i w_i \cdot X_i(e))$ 
  repeat
    for each  $i \in [0, m]$  do  $d_i \leftarrow 0$ 
    selezionare il batch  $B \subseteq Es$  di dim.  $b$ 
    for each esempio  $e \in B$  do
       $error \leftarrow pred(e) - Y(e)$ 
      for each  $i \in [0, m]$  do  $d_i \leftarrow d_i + error \cdot X_i(e)$ 
    for each  $i \in [0, m]$  do
       $w_i \leftarrow w_i - \eta * d_i / b$ 
  until terminazione
  return  $pred$ 

```

Le possibili *condizioni* di **terminazione** per l'algoritmo sono:

1. un numero massimo di passi (utile a evitare di divergere);
2. un errore vicino a 0;
3. ridotte variazioni di \bar{w} fra iterate successive.

Esempio — Classificatore per il problema dei post:

Classificatore corretto:

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

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

Si può leggere il modello come: $\widehat{Reads}(e)$ è vero (ossia il valore predetto per e risulta più vicino a 1 che a 0) sse $\text{Short}(e)$ è vero ed è anche vero $\text{New}(e)$ o $\text{Known}(e)$.

Tale funzione è analoga a quella appresa attraverso gli alberi di decisione.

Sono possibili alcune *varianti*: regolando le dimensioni dei lotti/batch, con b più piccole l'apprendimento risulta più veloce dato che si richiedono meno esempi per l'aggiornamento (ma si potrebbe compromettere la convergenza verso il minimo locale); considerando, invece, tutti gli esempi, si termina quando tutte le d_i sono nulle: tipicamente si parte con batch piccoli e si aumentare b fino alla convergenza (o al raggiungimento di prestazioni soddisfacenti).

Nella **discesa di gradiente incrementale** (**ONLINE GD**) l'aggiornamento dei pesi va fatto *dopo ogni esempio*, quindi, in tal caso, $b = 1$. Non si devono memorizzare i d_i , i pesi sono aggiornati direttamente. Ciò torna utile per esempi acquisiti in *streaming*. Nel caso di esempi non selezionati casualmente, si corre il rischio della cosiddetta **dimenticanza catastrofica**, ossia dell'adattamento ai dati più nuovi, dimenticando quelli meno recenti.

Separabilità Lineare

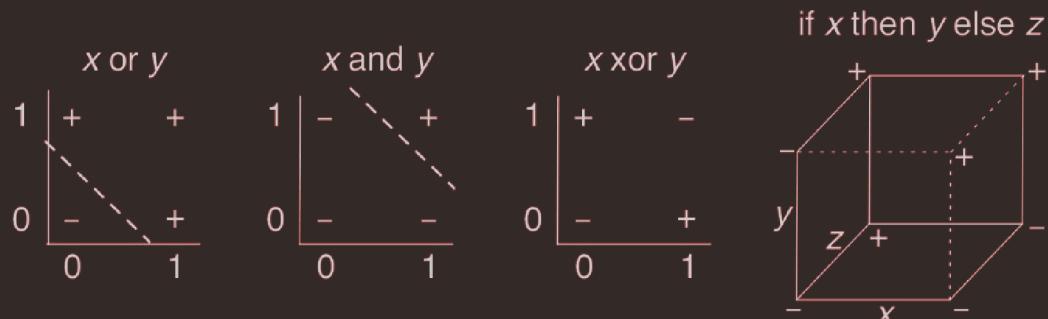
In uno spazio m -dimensionale, i.e. con m feature (di input), si definisce **iperpiano** l'insieme di punti che soddisfano un vincolo espresso come equazione *lineare* sulle variabili. Esso determina uno spazio di $(m - 1)$ dimensioni: ad esempio nel caso bidimensionale si tratta di una retta, mentre in quello tridimensionale sarà un piano, ecc.

Un **problema** di classificazione binaria si dice **linearmente separabile** quando esiste un iperpiano che separa gli esempi delle diverse classi. Un problema siffatto può essere risolto con la regressione logistica (usando un algoritmo di **GD STOCASTICA**): l'errore è riducibile arbitrariamente se il problema linearmente separabile. L'iperpiano separatore sarà l'insieme di punti per i quali $\sum_i w_i \cdot X_i = 0$, dove w contiene i coefficienti appresi. Dato un nuovo esempio, per la predizione della classe il modello avrà un valore maggiore di **0.5** da un lato e minore dall'altro.

Esempio — Separabilità lineare

Risultano problemi *linearmente separabili* quelli legati alle funzioni **or** e **and** (esempi rappresentati come coppie di bit, eventualmente anche con l'aggiunta di *rumore*).

Sono, invece *non linearmente separabili* il problema dello **xor** (*or esclusivo*), dato che nessuna retta può separare le due classi di esempi, ossia la funzione non è rappresentabile con un classificatore lineare, e il problema della funzione condizionale **if x then y else z** su feature di input booleane x, y, z



La separabilità lineare spesso non può essere determinata a priori facilmente.

Esempio — Problema su variabili booleane con target t vero se x vera e y vera, o se x falsa e z vera:

- risulta non separabile linearmente: $t = \text{if } x \text{ then } y \text{ else } z$;
- usando la **GD**, dopo 1000 epoche con $\eta = 0.05$, si ottiene un modello con i pesi seguenti (approssimati a 2 cifre decimali):

$$\text{lin}(e) = -0.1 \cdot x(e) + 4.06 \cdot y(e) + 4.06 \cdot z(e) - 3.98$$

da cui $\hat{t}(e) = \text{sigmoid}(\text{lin}(e))$

x	y	z	t	lin	\hat{t}
0	0	0	0	-3.98	0.02
0	0	1	1	0.08	0.52
0	1	0	0	0.08	0.52
0	1	1	1	4.14	0.98
1	0	0	0	-4.10	0.02
1	0	1	0	-0.04	0.49
1	1	0	1	-0.04	0.49
1	1	1	1	4.14	0.98

- solo per quattro esempi si hanno predizioni corrette, per gli altri si hanno valori vicini a 0.5;
- il modello risultante è abbastanza stabile anche ripetendo l'algoritmo con diverse inizializzazioni;
- l'incremento del numero di interazioni avvicina le predizioni \hat{t} a 0, 1, o 0.5.

Feature Target Categoriche

Quando Y è categorica con più di due / classi, essa può essere rappresentata usando variabili indicatrici binarie da imparare separatamente. Essendoci solo un valore di Y per ogni esempio, il modello dovrebbe essere capace di predire probabilità per ciascuna variabile indicatrice, con somma unitaria. È possibile apprendere modelli per tutte tranne una e ricavare la probabilità del suo valore dalle altre. Ciò introduce una forma di *asimmetria* che può risultare problematica con predizione debole del valore (a volte persino con probabilità negative!) per l'accumularsi di errori sulle altre variabili.

In alternativa si può considerare, per ogni valore, l'esponenziale d'una funzione lineare, da normalizzare. Si avranno così più parametri del necessario nel modello (*sovraparametrizzazione*), ma anche uno stesso trattamento per tutti i valori.

Data Y categorica con dominio rappresentato dalle tuple $\langle v_1, \dots, v_k \rangle$, si definisce la **funzione softmax** $\langle \alpha_1, \dots, \alpha_k \rangle \in \mathbb{R}^k \mapsto \mathbb{R}^k$ che calcola un vettore la cui i -esima componente sarà:

$$\text{softmax}(\langle \alpha_1, \dots, \alpha_k \rangle)_i = \frac{\exp(\alpha_i)}{\sum_{j=1}^k \exp(\alpha_j)}$$

Tale funzione assicura la positività e la somma unitaria potendo così definire una distribuzione di probabilità. Essa è correlata con la sigmoide, essendo:

$$\text{sigmoid}(x) = \frac{1}{\exp(-x) + 1} \stackrel{(*)}{=} \frac{\exp(x)}{\exp(0) + \exp(x)} = \text{softmax}(\langle 0, x \rangle)_2$$

la coppia $(0, x)$ corrisponde ai valori $(\text{false}, \text{true})$; $\text{softmax}(\langle 0, x \rangle)_2$ è la seconda componente della coppia risultante dalla softmax; si ha l'uguaglianza $(*)$ moltiplicando numeratore e denominatore per $\exp(x)$ quindi equivale al caso in cui la prima componente (per false) sia fissata a 0. Si noti come la softmax, come la sigmoide, *non* può rappresentare probabilità nulle.

La **regressione softmax** è una generalizzazione della regressione logistica (detta anche *regressione logistica/logit multinomiale*): si ha un'equazione lineare per ogni valore di Y , dove $\text{dom}(Y) = \{v_1, \dots, v_k\}$. Per la predizione per un esempio e e $\text{softmax}(\langle u_1(e), \dots, u_k(e) \rangle)$ calcola una tupla di k valori con j -esima componente corrispondente alla predizione per $Y = v_j$ e $u_j(e) = w_{0,j} + X_1(e) \cdot w_{1,j} + \dots + X_m(e) \cdot w_{m,j}$.

Il modello viene tipicamente ottimizzato usando la log-loss categorica: sia w_{ij} per X_i e v_j , e sia $Y(e) = v_q$:

$$\begin{aligned} \frac{\partial}{\partial w_{ij}} \text{logloss}(\text{softmax}(\langle u_1(e), \dots, u_k(e) \rangle), v_q) &= \frac{\partial}{\partial w_{ij}} - \log \left(\frac{\exp(u_q(e))}{\sum_j \exp(u_j(e))} \right) \\ &= \frac{\partial}{\partial w_{ij}} (\log (\sum_j \exp(u_j(e))) - u_q(e)) \\ &= [(\hat{Y}(e))_j - \mathbf{1}(j = q)] \cdot X_i \end{aligned}$$

con $\mathbf{1}(j = q) = 1$ se j è l'indice del valore osservato v_q e $(\hat{Y}(e))_j$ j-esima componente della predizione, valore predetto meno quello effettivo.

Nell'implementazione del modello vi sono problematiche connesse con la rappresentazione dei numeri:

- l'esponenziale di numeri grandi potrebbe non essere rappresentabile portando a un caso di *overflow*, ad esempio $\exp(800)$;
- l'esponenziale di numeri grandi negativi tende a **0**, comportando possibili casi di *underflow*, ad esempio $\exp(-800)$;

Osservato che l'aggiunta di una costante a ogni α_i nella softmax non ne cambia il valore, per la prevenzione di casi di over-/underflow, conviene sottrarre il massimo da ciascun valore per cui ci sarà sempre uno 0 e il resto dei valori risulteranno negativi. Su GPU e HW paralleli a bassa precisione le correzioni sono importanti.

Per grandi numeri di valori possibili il calcolo del denominatore diventa costoso. Si può ricorrere a una modalità di predizione in termini di un albero binario di valori, definita **softmax gerarchica**, simile alla softmax, ma più efficiente per grandi domini. Ad esempio in NLP può essere necessario predire prossima parola in un testo (fra milioni di possibilità).

Creazione di Feature di Input

Nella regressione lineare e in quella logistica, si considerano feature di input numeriche. In caso di altri tipi si devono fare delle trasformazioni:

- feature *categoriche*: vanno convertite in feature con dominio $\{0, 1\}$, tramite **one-hot encoding**, usando variabili-indicatrici;
- feature *a valori reali*: si possono usare direttamente in modelli lineari per \mathbf{Y} , una volta aggiustate le altre feature di input; spesso si adottano anche trasformazioni per crearne di nuove, dotate di proprietà desiderabili;
- feature *ordinali*, comprese quelle a valori reali: si possono considerare dei *tagli* per definire feature booleane, ad esempio da x , scegliendo un valore v del suo dominio, si considera una nuova feature, vera se $x > v$, oppure $x - v > 0$; nel *binning* si sceglie un insieme di soglie, $\alpha_1 < \alpha_2 < \dots < \alpha_k$, ricavando una feature booleana per ogni intervallo fra α_i e α_{i+1} , come per una funzione costante a pezzi; una feature può essere anche derivata da $\max(x - v, 0)$, funzione alla base delle *rectified linear unit* (ReLU) (cfr. Cap.8 di [PM23])

La progettazione di feature appropriate ai diversi problemi e relativi modelli è il tema principale della cosiddetta **feature engineering**. Spesso risulta difficile progettare buone feature. In seguito si considererà il problema aggiuntivo dell'apprendimento delle feature il quale è alla base del cosiddetto **representation learning** (cfr. Cap.8 di [PM23]).

4 Sovradattamento

Il **sovradattamento** (*overfitting*) è il problema che si manifesta con l'apprendimento di modelli basati su *regolarità apparenti* presenti nei dati di training, ma non in quelli di test o, in generale, nel mondo da cui essi sono tratti. Il fenomeno è tipico nella ricerca di *segnali* in dati distribuiti casualmente: si tratta di correlazioni *spurie* nei dati che non si riflettono sull'intero dominio del problema; si verifica un *eccesso di fiducia* nelle predizioni del modello rispetto a quanto autorizzino i dati a disposizione.

Vediamo un primo esempio:

Esempio — Sito con giudizi su ristoranti: da 1 a 5 stelle.

Si vogliono riconoscere e mettere in evidenza i ristoranti *eccellenti*: conviene usare il giudizio medio?

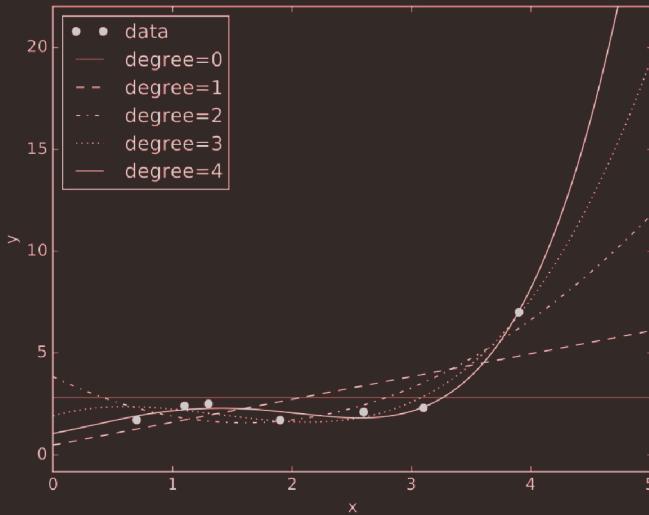
- per ristoranti con *molti* giudizi risulta molto difficile avere una media eccellente (da 5 stelle): richiede l'unanimità;
- non è difficile osservare un ristorante con *un solo* giudizio pari a 5 stelle: apparentemente rientrerebbe tra quelli eccellenti (con media di 5 stelle) ma è improbabile che lo sia davvero; analogo problema si manifesterebbe con ristoranti con uno o pochi giudizi molto bassi.

Previsioni estreme possono diventare problematiche con la disponibilità di nuovi dati. Questa forma di sovradattamento è legata alla cosiddetta **regressione verso la media**. I valori delle osservazioni (ad esempio i giudizi sui ristoranti) sono determinati da *qualità* e *caso*: più dati abbassano l'impatto della casualità.

Il sovradattamento può dipendere anche legato dalla *complessità del modello* scelto: un modello più complesso, con più parametri, risulta *più facile* da adattare ai dati di training rispetto a uno più semplice. Si può considerare il *caso estremo* in cui si preveda un modello con un parametro per ciascun esempio. Tuttavia esso andrà giudicato sulla base di nuovi esempi di test, non utilizzati per l'addestramento.

Esempio — Polinomio di grado k :

$$y = \sum_{i=1}^k w_i \cdot x^i$$



Con un algoritmo per la *regressione lineare* come **GD** minimizzando l'*errore quadratico* di un modello con y feature target, con le feature di input $1, x, x^2, \dots, x^k$ e vettore di parametri \vec{w} . Si considerano polinomi fino al grado $k = 4$ per il problema di regressione visto in precedenza.

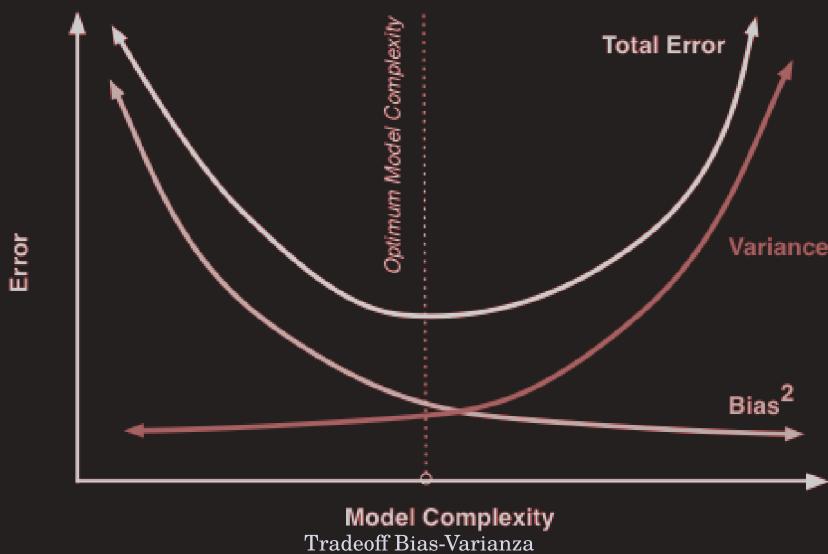
Polinomi di grado *maggiore* sarebbero più adattabili ai dati ma non necessariamente migliori: fanno predizioni più *estreme* nei casi di estrapolazione in quanto tendono asintoticamente verso $\pm\infty$ (tranne il modello costante). Il massimo k per cui $w_k \neq 0$ essendo pari o dispari porta, per x che tende a $\pm\infty$, a predizioni con lo stesso segno ovvero segno opposto. Ad esempio, un polinomio di grado 4 tende a $-\infty$ al diminuire di x : bisogna chiedersi se sia ciò ragionevole per il problema da risolvere.

Errore sul Test Set (cfr. [HTF09])

Dalla Statistica:

$$\text{Errore}_2(\text{Test}) = \text{Bias}^2 + \text{Varianza} + \text{Errore_irriducibile}$$

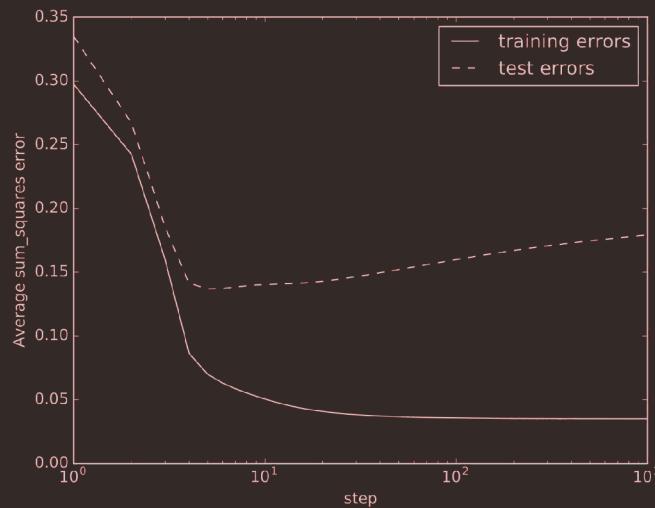
- **bias**: errore dovuto all'*imperfezione* del modello appreso, risulta basso quando esso ha una complessità vicina a quella ideale (**ground truth**) del *processo reale* che governa la generazione dei dati (cfr. figura seguente). Può essere scomposto in *bias di rappresentazione*, la differenza tra tipo di modello scelto e *ground truth*, e *bias di ricerca*, influenza dell'algoritmo che non cerca esaustivamente nello spazio di ipotesi.



Esempi

- alberi di decisione: il bias di rappresentazione è basso (si può rappresentare qualunque funzione, per parti) ma il bias di ricerca cresce con il numero di feature (moltissimi alberi);
- regressione lineare: se risolta analiticamente, si ha un bias di rappresentazione alto mentre il bias di ricerca è nullo per la soluzione analitica (maggiore se si usa la **GD**).
- **varianza**: errore dovuto all'insufficienza di dati disponibili; un modello più complesso (con più parametri) richiede più dati per cui, se il quantitativo di dati è fisso, si deve trovare un **compromesso** fra *bias* e *varianza*. Per un modello complesso, se non è disponibile il quantitativo di dati necessario allora il bias sarà basso e la varianza alta; per un modello semplice, se i dati disponibili sono sufficienti alla stima dei parametri si ha una bassa varianza ma un bias alto (cfr. figura precedente).
- **rumore**: errore *intrinseco* dovuto a dati che dipendono da *feature non considerate* nel modello e alla *naturale aleatorietà* nel processo di generazione dei dati.

Esempio — Errori (quadratico) nelle numero di iterazioni nella GD



- 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 esempi di training si tende ad avere più fiducia in un modello imperfetto, per cui l'errore sul test set aumenta

4.1 Pseudoconteggi

Come visto in precedenza, nella predizione il calcolo della *media* costituisce spesso una scelta ottimale rispetto a molte misure basate su dati di training. Ma la *media empirica* su *Es* non è sempre valida per stime riguardanti nuove istanze. Ma, in mancanza di dati a sufficienza, si può verificare il caso di un valore fra quelli del dominio mai osservato in *Es* avrebbe media nulla, così come la sua frequenza relativa se si deve stimare la probabilità. Ad esempio, si supponga di dover predire il voto di uno studente. La media risulta *appropriata* solo se sono stati già sostenuti diversi esami, mentre risulterebbe *non appropriata* se fosse basata su un solo esame o addirittura *indefinita* in assenza di esami.

Per porre rimedio a tali problemi si possono considerare *pseudo-esempi* aggiuntivi come *conoscenza pregressa* addizionale da usarsi per calcolare i cosiddetti **pseudoconteggi** (*prior counts*) di partenza. Essi saranno non negativi ma non necessariamente interi.

Disponendo dei valori osservati v_1, \dots, v_n , per predire il successivo \hat{v} , si assume di usare una predizione media a_0 , ottenuta da conoscenza pregressa (il parere dell'esperto), prima ancora di avere dati a disposizione, ossia quando $n = 0$; per i casi successivi si stimerà la media come segue:

$$\hat{v} = \frac{c \cdot a_0 + \sum_i v_i}{c + n}$$

con $c \in \mathbb{R}_+$ numero di dati fintizi assunti, *pseudo-esempi*, ciascuno con valore a_0 (non necessariamente intero). Quando $c = 0$ la predizione corrisponde a quella della media empirica, non utilizzabile senza dati osservati a disposizione ($n = 0$). Il valore di c regola l'importanza relativa fra ipotesi iniziale (*a priori*) e dati osservati.

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

Si assume che nuovi dati somiglieranno ai dati già acquisiti: un nuovo ristorante dovrebbe essere simile a quello medio (non sempre questa si rivela una buona assunzione); prima di raccogliere giudizi, è ragionevole adottare un giudizio medio a_0 fornito da un esperto.

Volendo fare predizioni accurate su ristoranti di *fascia top*, per stimare c si può considerare un ristorante in tale fascia con un *unico* giudizio di valore 5: esso dovrà essere simile agli altri della stessa fascia, in cui si suppone di avere a^t come giudizio medio (pesato sul numero di giudizi di ciascun ristorante).

Affinché tale ristorante confermi la media a^t della fascia top, per $n = 1$ ovviamente sarà $\sum_{i=1}^n v_i = 5$, quindi si avrà $a^t = \frac{c \cdot a_0 + 5}{c+1}$ da cui $c = \frac{5-a^t}{a^t-a_0}$.

Supponendo che $a_0 = 3$ e $a^t = 4.5$ allora $c = \frac{1}{3}$; invece, se fosse $a^t = 3.5$ allora risulterebbe $c = 3$.

Per *esercizio* si provi a generare un dataset per una Y booleana. Si selezioni un numero a caso $p \in [0, 1]$, da considerare come *ground truth* per $P(Y = 1)$ e poi si generino n esempi di training con l'assunzione che $P(Y = 1) = p$, usando un generatore pseudocasuale (campionando uniformemente su $[0, 1[$) e adottando un taglio rispetto a p . Siano n_1 e $n_0 = n - n_1$, rispettivamente, le frequenze di casi con $Y = 1$ e $Y = 0$. Si generino anche esempi di test (ad es. 10) sempre in base allo stesso p .

Problema: derivare uno stimatore \hat{p} dalle frequenze n_0 e n_1 . Una buona stima \hat{p} dovrà minimizzare l'errore su casi di test; per farsi un'idea, si possono effettuare diverse ripetizioni, ad esempio per $n = 1, 2, 3, 4, 5, 10, 20, 100, 1000$.

Come visto in precedenza, adottando errore quadratico o log-loss, la media empirica fornisce una stima ottimale: $\hat{p} = n_1 / (n_0 + n_1)$. Tuttavia pur minimizzando errore/loss sul training set, non ci sono garanzie sul test set. Tipicamente, se uno dei valori non occorre mai nel training set, si registrerebbe una log-loss infinita.

In tali casi, si può ricorrere allo **smoothing additivo di Laplace**:

$$\hat{p} = \frac{n_1 + 1}{n_0 + n_1 + 2} = \frac{n_1 + 1}{n + 2}$$

Fra tutti gli estimatori, minimizza log-loss ed errore quadratico sul il test set. Si osservi come l'aggiustamento equivalga all'aggiunta di due pseudo-esempi, uno con $Y = 0$ e uno con $Y = 1$.

4.2 Regolarizzazione

Secondo il *Rasoio di Occam* si dovrebbero preferire modelli semplici a quelli complessi.

A tale scopo si può considerare l'uso di un **regolarizzatore**, ossia un termine da aggiungere alla misura di adattamento del modello ai dati (tipicamente un errore, una loss) in modo da *penalizzare* la complessità dell'ipotesi da apprendere.

Si dovrà pertanto scegliere un'ipotesi h che minimizzi

$$\left[\sum_e \text{loss}(\hat{Y}(e), Y(e)) \right] + \lambda \cdot \text{regularizer}(\hat{Y})$$

dove:

- $\text{loss}(\hat{Y}(e), Y(e))$ misura la qualità dell'adattamento su e ;
- $\text{regularizer}(\hat{Y})$ penalizza la complessità del modello;
- il **parametro di regolarizzazione** λ , media tra adattamento e semplicità; esso può servire anche a sommare quantità su scale diverse e può essere determinato in base a conoscenza pregressa o tramite *cross-validation* (cfr. sezione seguente).

Si osservi che con *molti* esempi la somma a sinistra tende a dominare sul regolarizzatore, mentre con *pochi* esempi il regolarizzatore ha maggiore effetto.

Il parametro di regolarizzazione è un esempio di iperparametro del modello. Formalmente un **iperparametro** è un parametro usato per definire cosa/come si ottimizza e viene scelto in base a conoscenza pregressa, esperienza con problemi simili, o anche via **tuning** tramite *cross-validation*.

Ad esempio, nel caso degli *alberi* di decisione (binari), una misura della *complessità* è il numero di test, il quale dipende a sua volta dal numero delle foglie. Nella minimizzazione della loss si può aggiungere una funzione delle dimensioni dell'albero T :

$$\left(\sum_{e \in Es} \text{loss}(\hat{Y}(e), Y(e)) \right) + \gamma \cdot |T|$$

dove $|T|$ è la misura sopra indicata: un nuovo nodo-test risulta *utile* se riduce l'errore di γ .

Nel caso di *modelli con parametri* a valori reali, un regolarizzatore L_2 penalizza la somma dei quadrati dei parametri. Ad esempio, nella cosiddetta **ridge regression** esso viene sommato all'*errore quadratico*:

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

Anche nella *regressione logistica*, si adotta un regolarizzatore L_2 puntando a minimizzare la seguente *log-loss*:

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

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

Nell'implementazione della *regolarizzazione* L_2 , si introduce il *passo*:

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

con $b/|Es|$ perché λ è relativo all'intero dataset ma l'aggiornamento va operato *su ciascun batch* di dimensione b ; è anche possibile regolarizzare *dopo ogni iterata* su tutti gli esempi. Si noti che il fattore $\eta\lambda b/|Es|$ cambia di rado per cui può essere pre-calcolato.

Un *regolarizzatore* L_1 si basa sulla somma dei valori assoluti dei parametri. Ad esempio, nel modello detto **LASSO** [HTF09] si aggiunge $\lambda(\sum_i |w_i|)$ all'*errore quadratico*. Usando la *log-loss*:

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

con

- derivata parziale del termine $\text{sign}(w_i) = w_i/|w_i| \in \{-1, +1\}$: non definita in 0 , ma non serve spostarsi da 0 si è già sul valore minimo;
- si implementa facendo tendere a 0 ogni parametro di una costante: diventa nullo nel caso si cambi il segno del parametro;
- nell'**iterative soft-thresholding**, **GD** per la regressione logistica, si ha:

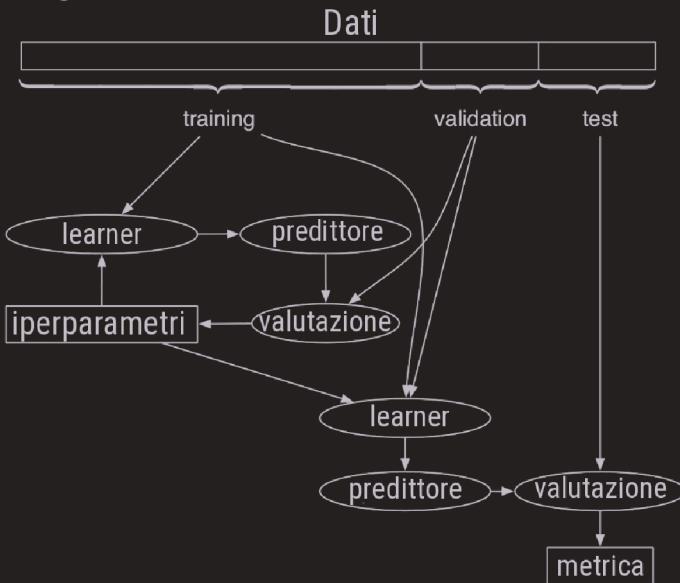
$$w_i \leftarrow \text{sign}(w_i) \cdot \max(0, |w_i| - \eta \cdot \lambda \cdot b/|Es|)$$

In caso di feature numerose, il regolarizzatore L_1 tende ad annullare molti pesi, facendo ignorare le relative feature, per cui diventa un meccanismo implicito per la **selezione delle feature**.

4.3 Cross Validation

Risulta difficile determinare la giusta complessità dei modelli da impiegare *prima* di aver visto i dati: essa dipende dal *numero* di parametri e/o da altri *iperparametri* dell'algoritmo.

A tale scopo si può adottare uno *schema generale* di apprendimento e valutazione, sintetizzato nella figura seguente:



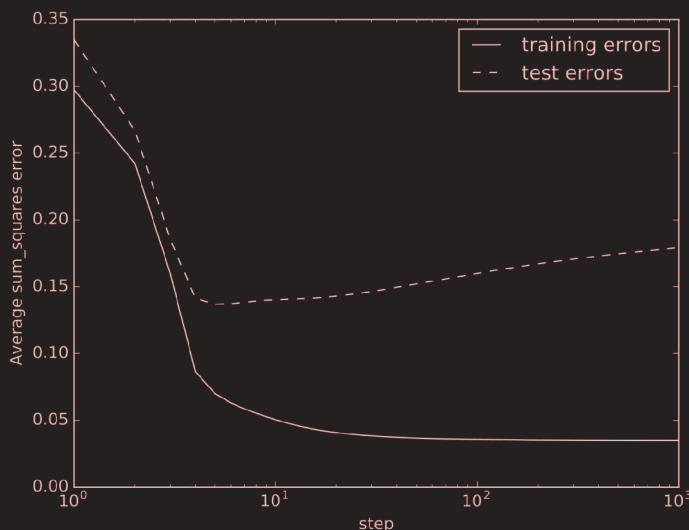
Si noti come il test set venga messo da parte per essere usato solo per la valutazione finale. Dev'essere composto da esempi *mai visti*, non utilizzati nella fase di addestramento, da impiegare come surrogato di nuovi esempi futuri.

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

Si dividono i dati *non di test* in due parti: un insieme di *training*, per l'apprendimento dei modelli e un insieme di **validazione** (*validation set*, *dev set*, *holdout*) per valutare diversi modelli (o diverse configurazioni degli iperparametri). Trovato il modello migliore, si userà il dataset (escluso il test set) per apprendere un modello.

La CV va ripetuta più volte scambiando i ruoli degli esempi per ottenere modelli diversi. Alla fine si sceglierà il modello migliore ossia quello con la migliore configurazione degli iperparametri.

Uso dell'errore di validazione: considerando il grafico precedente, l'errore sul training set decresce al crescere delle dimensioni dell'albero mentre l'errore sul validation set (e sul test set) si abbassa e poi prende a risalire.



La CV aiuta a scegliere la configurazione corrispondente al minimo dell'errore sul validation set. L'*ipotesi* è che si abbia lo stesso punto di minimo per anche per l'errore calcolato sul test set. Si può considerare come iperparametro da regolare anche il numero #passi di training prima di restituire un predittore, una forma di **early stopping**. Tipicamente l'addestramento ha bisogno di molti esempi per avere modelli migliori, ma ciò non deve andare a scapito del set per la validazione.

 Per confrontare modelli diversi si deve ripetere anche la CV esterna su diversi partizionamenti in training / test set. Ciò per evitare l'influenza di un fattore casuale come la scelta del test set che potrebbe favorire solo alcuni dei modelli considerati. Si confrontano i modelli sulla base delle medie delle misure calcolate su più test set e possibilmente anche della varianza (dev.standard) per valutare la *stabilità* dei modelli.

4.3.1 k-Fold Cross Validation

Disponendo solo di dataset limitati, si può puntare al riuso degli esempi per il training e la validazione per regolare i parametri (quindi la complessità) del modello appreso, scambiandone i ruoli:



Nella **k-Fold Cross Validation** i *passi* per valutare una scelta di parametri sono:

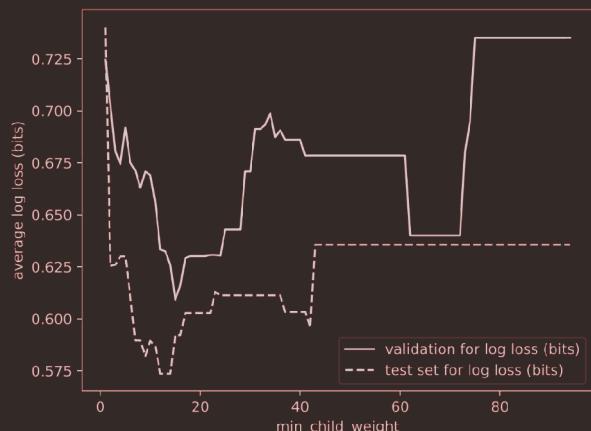
- si suddivide casualmente il dataset in **k** parti di uguali dimensioni, dette **fold**;
- si valutano i modelli con le diverse impostazioni: addestrando il modello **k** volte, ognuna con una distinta fold usata per la validazione;
- si determinano e restituiscono le impostazioni ottimali.

Ad esempio, ripetendo per **k = 10** volte, si addestrano modelli sul 90% e si valida sul 10% dei dati considerati. In tal modo la validazione coinvolge ogni esempio esattamente in *una* fold.

Esempio — Parametro degli alberi di decisione: *min_number_examples*.

Dal numero minimo di esempi per un nuovo nodo-figlio si ha un *criterio di stop*: se la soglia è troppo bassa si ha una tendenza al *sovradattamento*; se, invece, risulta troppo alta si ha una tendenza a *non generalizzare* abbastanza.

La seguente figura riporta il grafico delle log-loss su set di validazione di 5-fold CV:



Per la log-loss *media* sui set di validazione nelle 5 ripetizioni per valori crescenti del parametro si registra un minimo che rappresenta un valore *ottimale* per l'iperparametro. Dal grafico della log-loss sul test set si nota come esso sia ragionevole anche in base alle valutazioni sul test set.

4.3.2 Leave-One-Out Cross Validation

Nel caso estremo di un numero ancor più limitato di esempi, si può adottare una **k**-fold CV con **k** pari al numero di esempi di training, detta **Leave-One-Out Cross Validation**

Avendo **n** esempi di training, si apprendono **n** modelli: per ogni esempio **e**, si usano tutti gli altri per addestrare un modello da valutare infine su **e**.

La complessità cresce col numero degli esempi di training: il metodo diventa *poco pratico* con modelli prodotti in fasi di training *indipendenti* fra loro, mentre risulta adeguato se il modello creato in una esecuzione può essere *modificato* in modo efficiente nel successivo, sostituendo un solo esempio con un altro.

5 Modelli Compositi

5.1 Estensioni dei Modelli Lineari

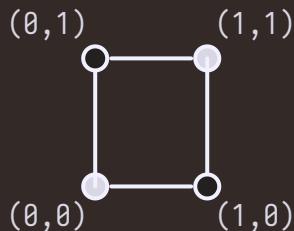
I modelli d'apprendimento visti fin qui forniscono le basi per altre tecniche di apprendimento. Sebbene qualsiasi funzione discreta possa essere rappresentata attraverso alberi di decisione, alcune funzioni, anche semplici, richiedono albero molto complessi. Anche i modelli lineari possono risultare spesso limitati, a meno che non si introducano meccanismi di costruzione di *nuove feature*.

Una possibile estensione considera l'uso di funzioni *non lineari* degli input originari in modo da creare nuovi input per i modelli lineari. La costruzione di nuove feature aumenta la *dimensionalità* dello spazio e funzioni che non erano lineari in origine possono diventarlo nel nuovo spazio o almeno rendere i problemi *linearmente separabili*.

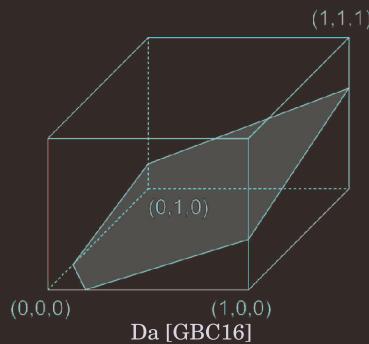
Esempio — la funzione **xor** risulta linearmente separabile nello spazio 3D con le feature x_1 , x_2 e x_1x_2 :

- $x_1x_2 = 1$ (*vero*) quando $x_1 = x_2 = 1$;
- $x = x_1$, $y = x_2$ e $z = x_1x_2$, ad esempio $\langle 1, 1 \rangle$ viene mappato su $\langle 1, 1, 1 \rangle$;
- per la *separazione* basta considerare un piano nel nuovo spazio.

Spazio 2D



Spazio 3D



Metodi basati su Kernel

Si possono definire modelli lineari su nuovi spazi di feature multidimensionali [Mit97, HTF09, Fla12, McK05]. Tali metodi si basano sulla trasformazione in nuovi problemi di ottimizzazione la cui soluzione dipende da una funzione **kernel**⁴ definita come prodotto nel nuovo spazio vettoriale:

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

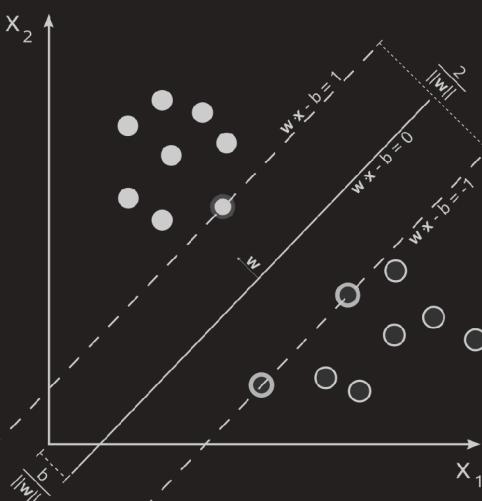
con $\phi : \mathbf{X} \rightarrow \mathbf{E}_S$ che rappresenta la trasformazione implicita / esplicita verso $\mathbf{E}_S = \mathbb{R}^n$, $n > m$, detto *embedding space*.

Tale approccio risulta utile in casi di non separabilità lineare nello spazio originario delle \mathbf{X} (ad esempio, considerando anche le feature x^2 e x^3 , si cercano polinomi di 3° grado). Il kernel generalizza il concetto di *prodotto vettoriale*, interpretabile come misura di *similarità* basata sulle nuove feature, ad esempio, prodotto di feature aggiuntive/sostitutive rispetto a quelle iniziali. Sono stati proposti molti tipi di kernel, anche per rappresentazioni delle istanze non vettoriali in origine, come ad esempio stringhe, liste, alberi, grafi, testi, distribuzioni di probabilità,...



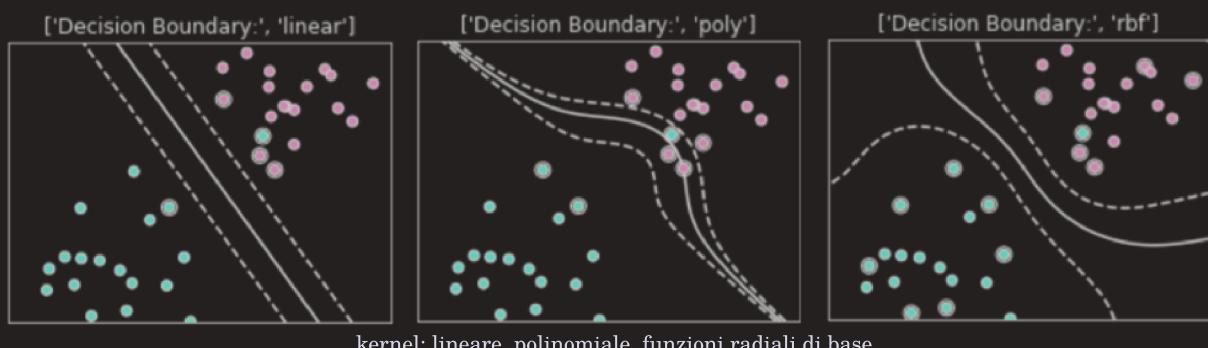
Con uno spazio delle feature esteso il problema del *sovradattamento* diventa più cogente

Una SVM è un classificatore basato su *kernel* [KM]: esso definisce un iperpiano, *superficie di decisione* ottimale, che divide esempi positivi dai negativi. In particolare si considera il *margine*, la distanza minima degli esempi dall'iperpiano: la SVM mira a trovare l'iperpiano di *massimo margine*. Gli esempi sul margine, che determinano tale iperpiano, si dicono vettori di *supporto*. La soluzione (parametri dell'iperpiano) sarà calcolata sulla base di prodotti applicati a tali vettori nello spazio di embedding (ossia tramite kernel). Il modello è *sparso* perché determinato da pochi vettori di *supporto*, i più vicini all'iperpiano, per cui, pur aumentando la dimensionalità, può evitare il sovradattamento.



SVM con margine unitario; vettori di supporto *cerchiati*

Kernel diversi (ovvero diverse funzioni ϕ) determinano diversi separatori:



Si veda la *live demo* @ Stanford.

È possibile usare modelli simili anche per risolvere problemi di regressione.

Ulteriori Estensioni

Reti neurali (cfr. cap. 8): ammettono input alla funzione lineare di appiattimento che sono a loro volta frutto di funzioni lineari compresse con coefficienti da determinare: con diversi layer siffatti si possono rappresentare *funzioni più complesse*.

Alberi di regressione: sono alberi con funzioni alle foglie, che possono costituire modelli lineari di regressione, potendo così rappresentare *funzioni lineari per parti*. Estensioni possono considerare foglie contenenti reti neurali o altri modelli (classificatori) non lineari. In fase di apprendimento, la scelta delle condizioni può essere fatta adottando un criterio di minimizzazione della varianza. Per la predizioni dell'output per un nuovo esempio, a partire dalla radice, lo si instrada giù per l'albero fino a individuare una foglia, quindi si userà il modello di regressione in essa contenuto.

5.2 Ensemble Learning

In generale, nell'**ensemble learning** si *combinano* le predizioni di un dato numero di modelli (semplici), ciascuno appreso da un learner *di base* a volte anche da parte del dataset. Ad esempio nelle *foreste* di alberi di decisione, le predizioni finali si otterranno attraverso medie ovvero meccanismi di voto. Vi sono diversi approcci di *ensemble learning* (*bagging*, *boosting*, *stacking*, ...) accomunati dall'essere costituiti da un certo numero di modelli-base addestrati sui dati e poi combinati attraverso meccanismi di voto o medie di tipo diverso.

5.2.1 Bagging: Random Forest

La **random forest** è un modello composto da più alberi di decisione. L'*idea* è quella di addestrare un certo numero di alberi su parti differenti del dataset. Per ogni esempio da classificare ognuno darà una predizione; le predizioni saranno si aggregate per formare la predizione finale per l'esempio dato.

L'efficacia dipende dalla *diversità* degli alberi generati in modo da ottenere da ciascuno predizioni anche diverse. A tale scopo per ogni albero si potrebbe considerare un sottoinsieme *casuale* di feature (ad esempio $1/m$ delle feature). In alternativa, le feature per i test nei nodi possono essere scelte in un *insieme ristretto*, variabile per ciascun albero (o nodo addirittura).

Nel **bagging** (o *bootstrap aggregating*) si considerano diversi sottoinsiemi degli esempi di training: dati m esempi di training ogni modello-base ne usa un piccolo numero. Ad esempio si possono considerare sottoinsiemi di m esempi estratti casualmente con rimpiazzo: in ognuno, risulteranno esempi assenti oppure anche duplicati il che porta a circa il 63.2% degli esempi originari per ciascun modello-base (come nel *bootstrap*). Nella *predizione finale* si dovrà mediare; ad esempio si può considerare la *classe* di maggioranza sulle predizioni ossia un meccanismo di *voto* basato sulla *moda*, che predica la classe più probabile / predetta da più modelli-base. Per foreste di alberi di regressione sono previste medie di tipo diverso.

5.2.2 Boosting

Nel **boosting** i modelli sono appresi in *sequenza* imparando dagli errori dei precedenti: si usa, in genere, lo stesso tipo di *learner di base* (ma si possono anche differenziare) quali, ad esempio, gli alberi di decisione o funzioni lineari compresse. Essi vanno via via adattati agli esempi sui quali il modello precedente commette errori: è come se gli esempi fossero *ordinati* in base agli errori. Come nel caso precedente, la *predizione finale* si ottiene aggregando le predizioni dei modelli prodotti nelle diverse iterate: somma, media pesata, moda...

Si osservi come i modelli-base possano anche essere *deboli*, non necessariamente ottimali, si chiamano infatti anche **weak learner**: essi devono giusto superare le prestazioni di un preditore casuale. L'ensemble risultante avrà prestazioni migliori garantite rispetto a ciascuno di essi.

Fra i più noti algoritmi **ADABoost** (*ADaptive BOOSTing*).

Functional Gradient Boosting

Il **functional gradient boosting** prevede un iperparametro K che corrisponde al numero di iterate da ripetere e quindi di modelli-base di regressione da apprendere. La *predizione finale* viene operata in funzione degli input nel modo seguente:

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

dove p_0 rappresenta la predizione iniziale, ad esempio la media, e d_i è la differenza dalla predizione precedente.

L' i -esima predizione si può scrivere:

$$p_i(X) = p_0 + d_1(X) + \cdots + d_i(X) = p_{i-1}(X) + d_i(X)$$

dove d_i viene costruito dal weak learner minimizzando l'errore di p_i (fissato p_{i-1}):

$$\sum_e \text{loss}(p_i(e), Y(e)) = \sum_e \text{loss}(\hat{d}_i(e), Y(e) - p_{i-1}(e))$$

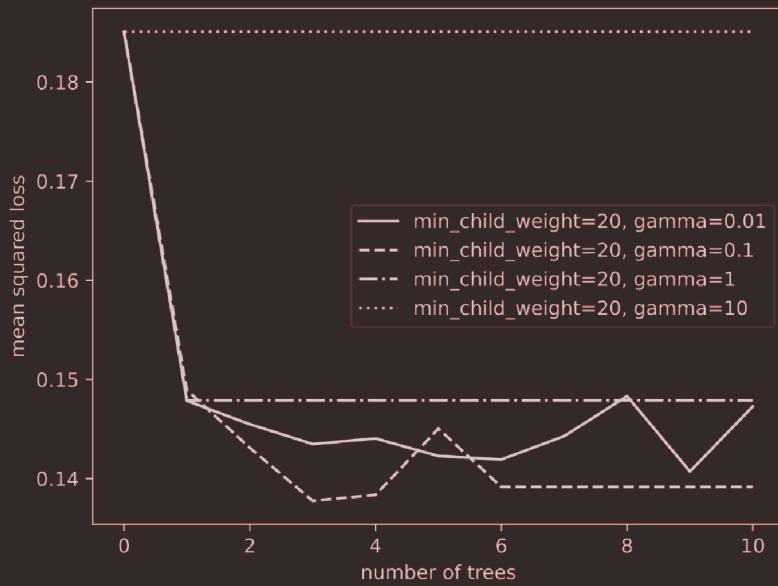
usando una qualsiasi loss, anche una per target continui, ma non la log-loss; $\hat{d}_i(e)$ deve ottimizzare $Y_i(e) - p_{i-1}(e)$. Si può operare come con un dataset con output modificati sottraendo le predizioni precedenti: ogni modello tende a correggere gli errori dei precedenti.

```
procedure Boosting_learner(Xs, Y, Es, L, K)
    Input
        Xs: insieme di feature di input
        Y: feature obiettivo
        Es: insieme di esempi di training
        L: learner di base
        K: numero di componenti dell'ensemble
    Output
        funzione per fare predizioni sugli esempi
    media  $\leftarrow \sum_{e \in Es} \frac{Y(e)}{|Es|}$ 
    define  $p_0(e) = \text{media}$ 
    for each  $i = 1$  to  $K$  do
         $E_i \leftarrow \{\langle Xs(e), Y(e) - p_{i-1}(e) \rangle \mid e \in Es\}$ 
         $d_i \leftarrow L(E_i)$ 
        define  $p_i(e) = p_{i-1}(e) + d_i(e)$ 
    return  $p_K$ 
```

con:

- p_i è la funzione di predizione sugli esempi;
- E_i è un nuovo insieme di esempi, generabile anche on demand: per ogni $e \in Es$, l'ultima predizione, $p_{i-1}(e)$, viene sottratta da $Y(e)$;
- d_i si calcola applicando il modello-base a E_i .

Esempio — Plot dell'errore quadratico sul validation set per K crescente:



Grafici per scelta dell'iperparametro γ fissato `minimum_child_size` (per 0 alberi, si usa l'errore per la predizione della media);

- per $\gamma = 10$ si hanno alberi degeneri, predizione della media;
- per $\gamma = 1$ gli alberi successivi al primo non estraggono info utile; errori simili a quello di un albero singolo;
- per $\gamma = 0.1, 0.01$, il boosting migliora le predizioni (si veda [AI Python](#)).

Gradient-Boosted Trees

Il **gradient-boosted tree** è un modello lineare con alberi binari come feature, appresi in sequenza tramite boosting (cfr. [PM23]).

Per la *regressione*, la predizione per $e = (\mathbf{x}_e, y_e)$ è data da

$$\hat{y}_e = \sum_{k=1}^K f_k(\mathbf{x}_e)$$

dove f_k è un albero rappresentato da un vettore di pesi \mathbf{w} , uno per foglia tale che $|\mathbf{w}|: \# \text{foglie}$ (ossia $1 + \#\text{test}$); q è la funzione che associa la feature di input \mathbf{x} alla foglia corrispondente implementando la struttura condizionale dell'albero, per cui $\mathbf{w}_{q(\mathbf{x})}$ rappresenterà il valore di un albero applicato a un esempio. Come loss si usa un errore quadratico regolarizzato

$$\left(\sum_e (\hat{y}_e - y_e)^2 \right) + \sum_{k=1}^K \Omega(f_k)$$

con $\Omega(f) = \gamma \cdot |\mathbf{w}| + \frac{1}{2} \lambda \cdot \sum_j w_j^2$.

Per la *classificazione*, la predizione può essere operata con una sigmoide (o una softmax) applicata alla somma degli alberi:

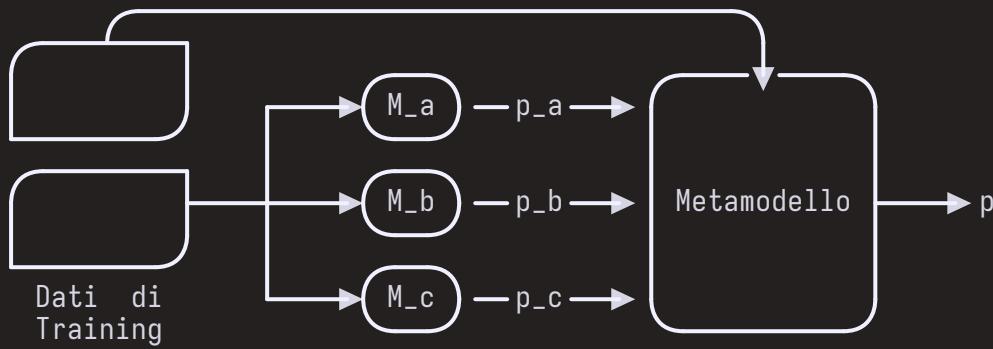
$$\hat{y}_e = \text{sigmoid}\left(\sum_{k=1}^K f_k(\mathbf{x}_e)\right)$$

La funzione da ottimizzare è:

$$\left(\sum_e \text{logloss}(\hat{y}_e, y_e) \right) + \sum_{k=1}^K \Omega(f_k)$$

Stacking (cenni)

Lo **stacking**, o *stacked generalization*, è un modello ensemble [Mit97, HTF09, Fla12] costituito da *modelli-base* di tipi diversi, come ad esempio alberi, modelli lineari, SVM, ecc. e un *meta-learner* che apprende come aggregare al meglio le decisioni [Stacking]. Uno schema generale è rappresentato in figura:



Riferimenti Bibliografici

- [PM23] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. 3a ed. 2023 (ch.7)
- [Fla12] P. Flach: *Machine Learning*. Cambridge University Press. 2012
- [HTF09] T. Hastie, R. Tibshirani, J. Friedman: *The Elements of Statistical Learning*. Springer. 2nd ed [ESL] 2009
- [McK05] D.J.C. MacKay: *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. 2005. disponibile su [ITPRN]
- [Mit97] T. Mitchell: *Machine Learning*. McGraw Hill. 1997
- [Ros58] F. Rosenblatt: *The perceptron: a probabilistic model for information storage and organization in the brain* Psychological Review 65 (6), pp. 386-408. 1958

Approfondimenti

- [ESL] <https://web.stanford.edu/~hastie/ElemStatLearn/>
- [ITPRN] <http://www.inference.org.uk/itprnn/book.html>
- [KM] <http://www.kernel-machines.org> <https://kernelmethods.blogs.bristol.ac.uk/>
- [Stacking] https://en.wikipedia.org/wiki/Ensemble_learning#Stacking

Note

¹ predizione binaria puntuale *alternativa* ristretta ai valori **0** o **1**

² la probabilità della predizione (**0**/**1**) per un esempio è $p^a(1-p)^{1-a}$ (bernulliana)
quindi, essendo gli esempi indipendenti, la verosimiglianza sarà pari a $\prod_{e \in Es} \hat{Y}(e)^{Y(e)}(1 - \hat{Y}(e))^{1-Y(e)}$

³ generalizzabili ai casi n-ari (feature discrete) e multivariati (test su più feature di input)

⁴ Si veda [KM] e, inoltre, *kernel trick* su wikipedia, kernel e SVM

⁵ *Curse of dimensionality* su Quora, WP, Medium-1 Medium-2