Statistical methods for machine learning

Mauro Tellaroli

Indice

1 Introduzione

1.1 Definizioni fondamentali

La *data inference* è lo studio dei metodi che utilizzano i dati per predirre il futuro. Il *Machine Learning* è uno strumento potente che può essere usato per risolvere una grossa parte dei problemi di *data inference*, inclusi i seguenti:

- **Clustering**: raggruppare i *data points* in base alle loro similarità;
- **Prediction**: assegnare delle etichette (*label*) ai *data points*;
- **Generation**: generare nuovi *data points*;
- **Control**: eseguire una sequenza di azioni in un ambiente con l'obiettivo di massimizzare una nozione di utilità.

Con *data point* si intende una serie di informazioni legate ad un unico elemento; un'analogia può essere un *record* in un database.

Gli algoritmi che risolvono una *learning task* in base a dei dati già semanticamente etichettati lavorano in modalità ***supervised learning***. A etichettare i dati saranno delle persone o la natura. Un esempio dell'ultimo caso sono le previsioni del meteo. D'altra parte, gli algoritmi che utilizzano i dati senza la presenza di etichette lavorano in modalità ***unsupervised learning***.

In questo corso ci si focalizzerà sul *supervised learning* e la progettazione di sistemi di *machine learning* il cui obiettivo è apprendere dei **predittori**, ovvero funzioni che mappano i *data points* alla loro etichetta.

1.1.1 *Label set* $\mathcal{Y}$

Verrà usata $\mathcal{Y}$ per indicare il *label set*, ovvero l'insieme di tutte le possibili etichette di un *data point*. Le etichette potranno essere di due tipi differenti:

1. **Categoriche** ($\mathcal{Y} = \{\text{sport}, \text{politica}, \text{economia}\}$): si parlerà di problemi di **classificazione**;
2. **Numeriche** ($\mathcal{Y} \subseteq \mathbb{R}$): si parlerà di problemi di **regressione**.

È importante sottolineare come la reale differenza tra le due tipologie di etichetta sia il significato e non la sua rappresentazione in quanto, si potrà sempre codificare un'etichetta categorica in un numero.

A sottolineare ciò è il fatto che nella regressione l'errore è tipicamente una funzione della differenza $|y - \hat{y}|$, dove $\hat{y}$ è la predizione di y. Nella classificazione, invece, l'errore è tipicamente binario: predizione corretta ($\hat{y} = y$) o errata ($\hat{y} \neq y$).

Quando ci sono solo due possibili etichette ($|\mathcal{Y}| = 2$), si ha un **problema di classificazione binario** e, convenzionalmente, verrà usata una codifica numerica $\mathcal{Y} = \{-1, 1\}$.

1.1.2 *Loss function* ℓ

Come già visto precedentemente, si vuole misurare l'errore che un predittore commette su una determinata predizione. Per farlo si userà una **funzione di loss** ℓ non negativa che misurerà la discrepanza $\ell(y, \hat{y})$ tra l'etichetta predetta $\hat{y}$ e quella corretta y. Si assumerà sempre $\ell(y, \hat{y}) = 0$ quando $\hat{y} = y$.

La funzione di loss più semplice per la classificazione è la **zero-one loss**:

$$\ell(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{altrimenti} \end{cases}$$

Nella regressione, le tipiche funzioni di loss sono:

- la ***absolute loss***: $\ell(y, \hat{y}) = |y - \hat{y}|$

- la ***quadratic loss***: $\ell(y, \hat{y}) = (y - \hat{y})^2$

In alcuni casi può essere conveniente scegliere l'etichetta predetta da un insieme $\mathcal{Z}$ diverso da $\mathcal{Y}$. Per esempio, si consideri il problema di assegnare una probabilità $\hat{y} \in (0, 1)$ all'evento $y =$ "pioverà domani". In questo caso, $\mathcal{Y} = \{$"piove", "non piove"$\}$ e $\mathcal{Z} = (0, 1)$. Indicando questi due eventi con 1 (piove) e 0 (non piove), si può usare una funzione di loss per la regressione, come la *absolute loss*:

$$\ell(y, \hat{y}) = |y - \hat{y}| = \begin{cases} 1 - \hat{y} & y = 1 \quad \text{(piove)} \\ \hat{y} & y = 0 \quad \text{(non piove)} \end{cases}$$

Per penalizzare maggiormente le predizioni che distano troppo dalla realtà, si può usare una ***logarithmic loss***:

$$\ell(y, \hat{y}) = \begin{cases} \ln \frac{1}{\hat{y}} & y = 1 \quad \text{(piove)} \\ \ln \frac{1}{1-\hat{y}} & y = 0 \quad \text{(non piove)} \end{cases}$$

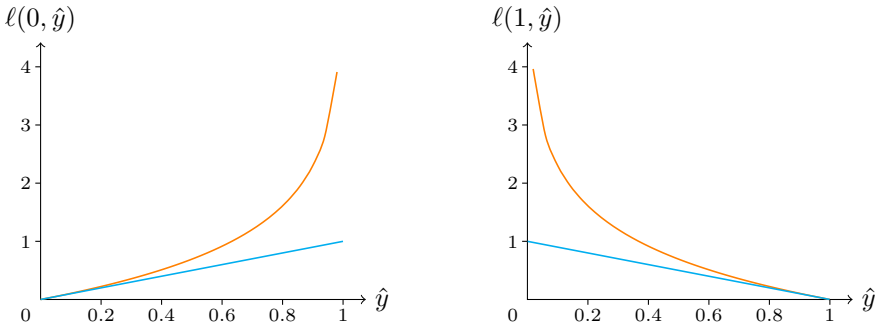

Figura 1: Confronto tra *absolute loss* e *logarithmic loss*; a sinistra il caso $y = 0$, a destra $y = 1$.

Si noti in figura 1 come la *logarithmic loss* tenda ad infinito quando la predizione è opposta all'etichetta reale:

$$\lim_{\hat{y} \to 1^-} \ell(0, \hat{y}) = \lim_{\hat{y} \to 0^+} \ell(1, \hat{y}) = +\infty$$

In pratica questo previene l'utilizzo di predizioni $\hat{y}$ troppo sicure, quindi troppo vicine a zero o uno.

1.1.3 *Data domain* $\mathcal{X}$

Verrà usata $\mathcal{X}$ per indicare l'insieme dei *data points*; ogni suo punto $x \in \mathcal{X}$ è tipicamente un record di un database formato da *feature*:

$$x = (x_1, \ldots, x_d)$$

Spesso un *data point* può essere codificato come un vettore i cui elementi sono le sue *feature*. Questa codifica risulta naturale in presenza di quantità omogenee, come i pixel di un'immagine o una lista di occorrenze di parole in un testo. Quando invece i dati presenti utilizzano unità di misura differenti, come "età" e "altezza", la codifica non risulta più immediata. Ci sarà bisogno di una procedura che codifichi i dati in modo da ottenere uno spazio vettoriale omogeneo e coerente con i dati iniziali.

In questo corso si assumerà che i dati possano essere rappresentati da vettori di numeri:

$$\mathcal{X} \equiv \mathbb{R}^d$$

1.1.4 Predittori f

Un **predittore** è una funzione $f : \mathcal{X} \to \mathcal{Y}$ che mappa i *data points* alle etichette (o $f : \mathcal{X} \to \mathcal{Z}$). Sì può quindi dire che in un problema di predizione l'obiettivo è ottenere una funzione f che genera delle predizioni $\hat{y} = f(x)$ tali che $\ell(y, \hat{y})$ sia basso per il maggior numero di punti $x \in \mathcal{X}$ osservati. In pratica, **la funzione f è definita da un certo numero di parametri in un dato modello**. Un esempio sono i parametri di una rete neurale.

1.1.5 Esempi

Nel *supervised learning* un **esempio** è una coppia (x, y) dove x è un *data point* e y la sua reale etichetta.

In alcuni casi x ha un'unica y, come nel caso in cui y rappresenta una proprietà oggettiva di x; in altri casi, invece, x può avere diverse y associate, come quando le y sono soggettivamente assegnate da persone.

1.1.6 *Test set* e *test error*

Per poter stimare la qualità di un predittore si usa un insieme di esempi detto ***test set***:

$$\{(x'_1, y'_1), \ldots, (x'_n, y'_n)\}$$

Data una *loss function* ℓ, il *test set* viene usato per calcolare il ***test error*** di un predittore f:

$$\frac{1}{n} \sum_{t=1}^{n} \ell(\underbrace{y'_t}_{\text{reale}}, \overbrace{f(x'_t)}^{\text{predetta}})$$

Il *test error* ha quindi lo scopo di calcolare la prestazione media del predittore su dei dati reali.

1.1.7 *Learning algorithm* A

Si definisce *training set* S un insieme di esempi:

$$S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$$

che viene usato dal ***learning algorithm*** A per produrre un predittore $A(S)$. Informalmente, il *learning algorithm* "impara" dal *training set*.

$$\underbrace{\{(x_1, y_1), \ldots, (x_m, y_m)\}}_{S} \longrightarrow \boxed{A \quad \boxed{\ell}} \longrightarrow A(S) = f : \mathcal{X} \to \mathcal{Y}$$

Il *test set* e il *training set* vengono solitamente prodotti assieme attraverso un processo di collezione dati e etichettamento. Dato l'insieme di esempi preparati, questo verrà partizionato in *test set* e *training set*, tipicamente tramite una divisione casuale. **Obiettivo del corso è lo sviluppo di una teoria che ci guidi nella progettazione di *learning algorithm* che generano predittori con un basso *test error*.**

1.1.8 *Training error* ℓ_S

Sia $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ il *training set*; viene definito, equivalentemente al *test error*, il ***training error***:

$$\ell_S(f) = \frac{1}{m} \sum_{t=1}^{m} \ell(y_t, f(x_t))$$

Un approccio intuitivo alla progettazione di *learning algorithm* è quello di assumere che il *training error* $\ell_S(f)$ del predittore f sia correlato con il suo *test error*.

1.2 Empirical Risk Minimization (ERM)

1.2.1 Definizione

Sia $\mathcal{F}$ un insieme di predittori e ℓ una *loss function*. L'*empirical risk minimizer* (ERM) è il *learning algorithm* A che restituisce un predittore in $\mathcal{F}$ che **minimizza il *training error***:

$$A(S) \in \operatorname*{argmin}_{f \in \mathcal{F}} \ell_S(f)$$

Si noti come $A(S)$ appartenga e non uguagli il minimo; questo perchè ci potrebbero essere più $f \in \mathcal{F}$ che minimizzano $\ell_S(f)$.

1.2.2 Predittori con *test error* elevato

Quando in $\mathcal{F}$ tutti i predittori hanno un *test error* alto, ERM produrrà un pessimo predittore. **Per trovare un buon predittore, ovvero un predittore con un *test error* basso, ci sarà quindi bisogno che $\mathcal{F}$ sia sufficientemente grande**.

Tuttavia, se $\mathcal{F}$ è troppo grande, anche in questo caso verrà prodotto un pessimo predittore. Un esempio è il seguente.

Si consideri il seguente problema "giocattolo":

$$\mathcal{Y} = \{-1, 1\} \qquad \mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$$

Si prenda l'insieme $\mathcal{F}$ contenente un classificatore $f : \mathcal{X} \to \mathcal{Y}$ per ognuna delle possibili combinazioni di etichettamento dei cinque *data points*. $\mathcal{F}$ sarà quindi formata da $2^5 = 32$ classificatori:

$$\mathcal{F} = \{f_1, \ldots, f_{32}\}$$

$\mathcal{F}$	$f(x_1)$	$f(x_2)$	$f(x_3)$	$f(x_4)$	$f(x_5)$
f_1	1	1	1	1	1
f_2	1	1	1	1	-1
f_3	1	1	1	-1	1
f_4	1	1	1	-1	-1
f_5	1	1	-1	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
f_{31}	-1	-1	-1	-1	1
f_{32}	-1	-1	-1	-1	-1

Si supponga che il *training set* S contenga solo tre *data points* qualsiasi e il *test set* contenga gli altri due. Sia f^* il predittore usato per etichettare i dati che quindi avrà zero *test* e *training error*; ogni etichetta y_t sarà quindi ottenuta da f^*:

$$y_t = f^*(x_t) \quad \forall t = 1, \ldots, 5$$

Per rendere l'idea, si prenda come esempio:

$$f^* = f_3$$

$$\begin{aligned} S &= \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} \\ &= \{(x_1, 1), (x_2, 1), (x_3, 1)\} \end{aligned}$$

Nonostante ad avere *test error* nullo sia solo f_3, ad avere il *training error* nullo sono i quattro classificatori che hanno $y_1, y_2, y_3 = 1$ ovvero f_1, f_2, f_3, f_4. Questo perchè il *training set* S contiene solo i primi 3 *data points*.

Siamo quindi nella situazione in cui ERM trova più predittori con ℓ_S minimo e non ha abbastanza informazioni per capire quale di questi sia migliore a livello di *test error*.

Il problema dell'esempio appena visto è che $\mathcal{F}$ è troppo grande rispetto al *training set*. La domanda che sorge spontanea è quindi: Quanto deve essere grande $\mathcal{F}$ per poter ottenere un buon predittore tramite ERM?

La teoria dell'informazione ci suggerisce che S debba avere cardinalità $\log_2 |\mathcal{F}|$ o, viceversa, $\mathcal{F}$ debba avere cardinalità 2^m. Quindi, nell'esempio di prima, il *training set* avrebbe dovuto contenere almeno $\log_2 |\mathcal{F}| = 5$ *data points*.

1.2.3 *Overfitting* e *underfitting*

I due eventi visti nella sezione precedente, che portano alla generazione di un predittore con *test set* elevato, vengono chiamati:

- ***Underfitting***: si verifica quando il *training error* è elevato;
- ***Overfitting***: si verifica quando il *training error* è basso ma il *test error* è alto.

Quando A è ERM e S ha dimensione fissata $|S| = m$:

- Ci si aspetta *overfitting* quando $\log_2 |\mathcal{F}| \gg m$;
- Ci si aspetta *underfitting* quando $\log_2 |\mathcal{F}| \ll m$.

1.2.4 Etichette rumorose

Il fenomeno dell'*overfitting* spesso accade quando le etichette sono rumorose, ovvero quando le etichette y non sono deterministicamente associate con i *data points* x. Questo può accadere per i seguenti motivi (non mutuamente esclusivi tra loro):

1. **Incertezza umana**: se ad etichettare S sono delle persone, ci sarà dell' incertezza in quanto persone diverse potrebbero avere opinioni diverse;
2. **Incertezza epistemica**: ogni *data point* è rappresentato da un vettore delle *feature* che non contiene abbastanza informazioni per determinare univocamente l'etichetta;
3. **Incertezza aleatoria**: il vettore delle *feature* che rappresenta il *data point* è ottenuto attraverso delle misurazioni rumorose.

Le etichette rumorose portano all'*overfitting* perchè possono ingannare l'algoritmo su quale sia la "vera" etichetta di una certo *data point*.

2 Gli algoritmi *Nearest Neighbor*

2.1 *Nearest Neighbor* (NN)

2.1.1 Definizione

Verrà introdotto ora l'algoritmo di ***Nearest Neighbor* (NN)** per la classificazione binaria con *feature* numeriche:

$$\mathcal{X} = \mathbb{R}^d \qquad \mathcal{Y} = \{-1, 1\}$$

NN non è un'istanza di ERM in quanto non punta a minimizzare ℓ_S.

L'idea di NN è la sueguente:

- **Predici ogni punto del *training set* con la propria etichetta;**
- **Predici gli altri punti con l'etichetta del punto del *training set* che è più vicino al punto interessato.**

Più formalmente, dato un *training set*:

$$S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$$

l'algoritmo A_{NN} genera un classificatore $h_{\text{NN}} : \mathbb{R} \to \{-1, 1\}$ definito come segue:

$$h_{\text{NN}}(x) = \text{etichetta } y_t \text{ del punto } x_t \in S \text{ più vicino a x}$$

Se a minimizzare la distanza con x sono più punti, si predirrà l'etichetta più presente tra i putni vicini. Se non c'è una maggioranza di etichette tra i punti più vicini si predirrà un valore di default $\in \{-1, 1\}$.

Presi due punti $x = (x_1, \ldots, x_d)$ e $x_t = (x_{t,1}, \ldots, x_{t,d})$, la distanza $||x - x_t||$ verrà calcolata tramite la distanza euclidea:

$$||x - x_t|| = \sqrt{\sum_{i=1}^{d} (x_i - x_{t,i})^2}$$

Ogni classificatore binario $f : \mathbb{R}^d \to \{-1, 1\}$ partiziona $\mathbb{R}^d$ in due regioni (come mostrato in figura 2):

$$\{x \in \mathbb{R}^d : f(x) = 1\} \quad , \quad \{x \in \mathbb{R}^d : f(x) = -1\}$$

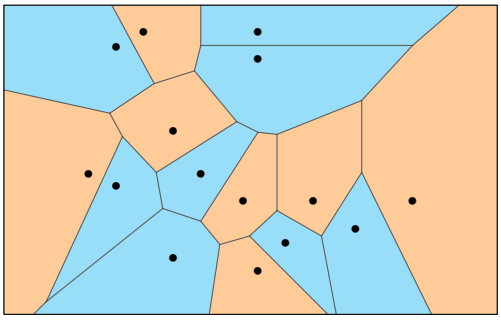

Figura 2: Diagramma di Voronoi in $\mathbb{R}^2$; tutti i punti x interni a una cella con centro $\bullet\, x_t$ sono tali che $h_{\text{NN}}(x) = y_t$

2.1.2 Efficienza ed efficacia

Siccome il funzionamento di NN implica la memorizzazione di tutto il *training set*, **l'algoritmo non scala bene con il numero di $|S| = m$ di *training point***. Inoltre, calcolare un qualsiasi $h_{\text{NN}}(x)$ è costoso, in quanto richiede di calcolare la distanza tra x e tutti gli altri punti di S; questo in $\mathbb{R}^d$ comporta un costo di $\Theta(dm)$.

Infine, si noti come, vista la completa memorizzazione di S, **NN generi sempre un classificatore h_{NN} con *training error* nullo**:

$$\ell_S(h_{\text{NN}}) = 0$$

2.2 $k-$Nearest Neighbor ($k-$NN)

2.2.1 Definizione

Partendo dagli algoritmi NN, si può ottenere una famiglia di algoritmi detta $k-$NN; il parametro k assume tipicamente i valori $k = 1, 3, 5, \dots$ con $k < |S|$.

Questi algoritmi sono definiti come segue: dato un *training set* S e un punto $x \in \mathcal{X}$, $k-$NN genererà un predittore $h_{k\text{-NN}}$ tale che:

$$h_{k\text{-NN}}(x) = \text{etichetta } y_t \text{ appartenente alla maggioranza dei } k \text{ punti più vicini a } x$$

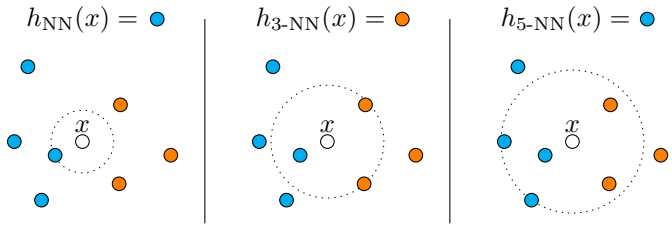

Figura 3: Esempi di $h_{k\text{-NN}}$ con $\mathcal{X} = \mathbb{R}^2$; si noti come, con lo stesso *training set*, la predizione cambia al variare di k.

2.2.2 Efficienza ed efficacia

A livello di efficienza $k-$NN soffre degli stessi problemi di NN vista la memorizzazione dell'intero *training set*.

Per quanto riguarda la sua efficacia invece, $k-$NN non ha sempre un *training error* nullo:

$$\ell_S(h_{k\text{-NN}}) \geq 0$$

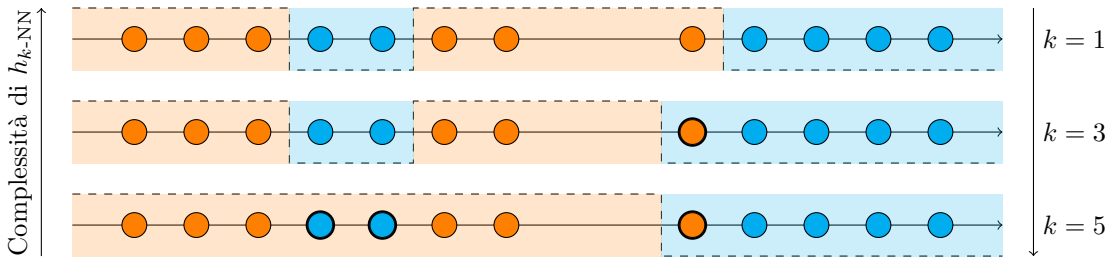

Figura 4: Esempi di $h_{k\text{-NN}}$ con $\mathcal{X} = \mathbb{R}$.

Come si può infatti notare dalla figura 4, nei casi con $k = 3$ e $k = 5$ sono presenti punti errati (evidenziati in grassetto) considerati dal classificatore come *outlier*. Inoltre **al crescere di k**

cresce anche la "semplicità" del classificatore così come il numero di punti errati. L'estremo di ciò è quando $k = |S|$; in questo caso infatti $h_{k\text{-NN}}$ diventa un classificatore costante che predice sempre l'etichetta più presente in tutto S.

In un generico classificatore $h_{k\text{-NN}}$ tipicamente succede che:

- **Se k è troppo basso** si ottiene un classificatore che si "fida" troppo del *training set*, ottenendo quindi ***overfitting***;
- **Se k è troppo alto**, si ottiene un classificatore troppo semplice, ottenendo ***underfitting***.

Tutti i classificatori introdotti fino ad'ora sono classificatori binari ($|\mathcal{Y}| = 2$). Tuttavia $k-$NNpuò essere usato anche per:

- problemi di classificazione multiclasse ($|\mathcal{Y}| > 2$): si opera come nel caso binario, predicendo quindi l'etichetta più presente nei k punti più vicini;
- problemi di regressione ($\mathcal{Y} = \mathbb{R}$): si predice la media aritmetice delle etichette dei k punti più vicini.

3 *Tree Predictors*

3.1 Definizione

Come già visto, mentre alcuni tipi di dato hanno una naturale rappresentazione vettoriale $x \in \mathbb{R}^d$, altri non ce l'hanno. Un esempio possono essere dei *record* medici, dove i dati contengono i seguenti campi:

$$\texttt{età} \in \{12, \dots, 90\}$$
$$\texttt{fumatore} \in \{\text{sì}, \text{no}, \text{ex}\}$$
$$\texttt{peso} \in [10, 200]$$
$$\texttt{sesso} \in \{\text{M}, \text{F}\}$$
$$\texttt{terapia} \in \{\text{antibiotici}, \text{cortisone}, \text{nessuna}\}$$

Anche convertendo questi tipi di dato in dati numerici, gli algoritmi basati sulla distanza euclidea, come il $k-$NN, potrebbero non andare molto bene.

Per poter applicare la *data inference* su dati le cui *feature* variano in insiemi eterogenei $\mathcal{X}_1, \dots, \mathcal{X}_d$, verrà introdotta una nuova famiglia di predittori: i *tree predictors*.

Un *tree predictor* è un albero ordinato e radicato dove ogni nodo può essere una **foglia** o un **nodo interno**. È importante sottolineare che in un albero ordinato i figli di ogni nodo sono anch'essi ordinati e quindi numerabili consecutivamente. In figura 5 viene mostrato un esempio di *tree predictor* binario le cui *feature* sono:

$$\texttt{previsione} \in \{\text{sole}, \text{nuvole}, \text{pioggia}\}$$
$$\texttt{umidità} \in [0, 100]$$
$$\texttt{vento} \in \{\text{sì}, \text{no}\}$$

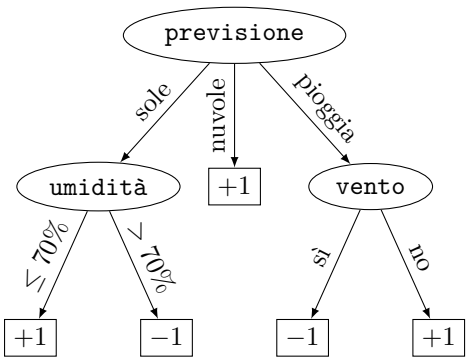

Figura 5: Esempio classico di *tree classifier* per una classificazione binaria.

Sia $\mathcal{X} = \mathcal{X}_1, \dots, \mathcal{X}_d$, dove ogni $\mathcal{X}_i$ rappresenta il dominio dell'i-esimo attributo (o *feature*) x_i. **Il *tree predictor* $h_T : \mathcal{X} \to \mathcal{Y}$ è un predittore definito da un albero T i cui nodi interni corrispondono a dei test e le cui foglie corrispondono a delle etichette $y \in \mathcal{Y}$.**

Un test su un attributo i su un nodo interno con k figli è una funzione $f : \mathcal{X} \to \{1, \dots, k\}$. f mappa ogni elemento di $\mathcal{X}_i$ a un nodo figlio. Due esempi possono essere i seguenti:

$$\mathcal{X}_i = \{a, b, c, d\} \qquad k = 3$$
$$f(x_i) = \begin{cases} 1 & x_i = c \\ 2 & x_i = d \\ 3 & x_i \in \{a, b\} \end{cases}$$

$$\mathcal{X}_i = [0, 100] \qquad k = 2$$
$$f(x_i) = \begin{cases} 1 & x_i \in [0, 70] \\ 2 & x_i \in (70, 100] \end{cases}$$

L'esempio di destra è riferito all'attributo `umidità` di figura 5.

La predizione $h_T(x)$ è calcolata come segue:

1. $v \leftarrow r$ (r è la radice di T)
2. se v è una foglia ℓ, si restituisce l'etichetta $y \in \mathcal{Y}$ associata a ℓ;
3. altrimenti, sia $f : \mathcal{X}_i \to \{1, \ldots, k\}$ il test associato a v, **assegna** $v \leftarrow v_j$ dove $j = f(x_i)$ e v_j indica il j-esimo figlio di v;
4. vai al punto 2.

Se $h_T(x)$ restituisce la foglia ℓ, si dirà che l'esempio x è indirizzato a ℓ.

3.2 Costruzione di un *tree predictor*

3.2.1 Idea generale

Dato un *training set* S, si vedrà ora come costruire un *tree predictor*. Per semplicità, si guarderà solo ad una classificazione binaria $\mathcal{Y} = \{-1, 1\}$ e verranno usati solo alberi binari completi, cioè alberi dove ogni nodo interno ha due figli.

L'idea è quella di far crescere l'albero partendo da un singolo nodo (che dovrà essere una foglia). L'etichetta di quest'unica foglia sarà l'etichetta $\hat{y} \in \mathcal{Y}$, ovvero l'etichetta più presente nel *training set*. Si avrà quindi inizialmente, un classificatore che assegna a tutti i *data point* l'etichetta $\hat{y}$. **L'albero sarà fatto crescere scegliendo una foglia e rimpiazzandola con un nodo interno e due nuove foglie**.

3.2.2 *Training error*

Si chiami T l'albero cresciuto fino a un certo punto e h_T il classificatore corrispondente. **Obiettivo è calcolare il contributo che ogni foglia dà al *training error* $\ell_S(h_T)$.**

Presa una foglia ℓ, si vuole capire che etichetta assegnarle per minimizzare ℓ_S.

Si definisca:

$$S_\ell = \{(x_t, y_t) \in S : x_t \text{ è indirizzato a } \ell\}$$

S_ℓ è quindi l'insieme degli esempi di *training* che sono indirizzati alla foglia ℓ. Si divida ora S_ℓ in due sottoinsiemi:

$$S_\ell^+ = \{(x_t, y_t) \in S_\ell : y_t = +1\}$$
$$S_\ell^- = \{(x_t, y_t) \in S_\ell : y_t = -1\}$$

Il primo conterrà tutti gli esempi di *training* che vengono indirizzati a ℓ con etichetta positiva mentre il secondo con etichetta negativa. Di questi insiemi si prenda il loro numero di elementi:

$$N_\ell^+ = |S_\ell^+| \qquad N_\ell^- = |S_\ell^-| \qquad N_\ell = |S_\ell|$$

È facile capire che se la maggior parte degli esempi di *training* che vengono indirizzati alla foglia ℓ hanno etichetta positiva, allora l'etichetta che bisognerà dare a ℓ, per minimizzare il suo errore ℓ_S, sarà l'etichetta positiva (chiaramente lo stesso discorso vale per l'etichetta negativa); questa intuizione può essere quindi usata per assegnare l'etichetta y_ℓ alla foglia ℓ nel seguente modo:

$$y_\ell = \begin{cases} +1 & N_\ell^+ \geq N_\ell^- \\ -1 & \text{altrimenti} \end{cases}$$

Di conseguenza la foglia ℓ sbaglierà la sua previsione su $\min\{N_\ell^+, N_\ell^-\}$ esempi di *training*. Per facilitare delle successive osservazioni moltiplichiamo e dividiamo per N_ℓ:

$$\min\{N_\ell^+, N_\ell^-\} = \min\left\{\frac{N_\ell^+}{N_\ell}, \frac{N_\ell^-}{N_\ell}\right\} N_\ell$$

Quindi se il valore appena scritto è l'errore che una singola foglia ℓ fa, il *training error* sarà:

$$\ell_S(h_T) = \frac{1}{m}\sum_{\ell} \min\left\{\frac{N_\ell^+}{N_\ell}, \frac{N_\ell^-}{N_\ell}\right\} N_\ell$$
$$= \frac{1}{m}\sum_{\ell} \psi\left(\frac{N_\ell^+}{N_\ell}\right) N_\ell$$

Dove viene introdotta la funzione ψ, definita in $[0,1]$:

$$\psi(a) = \min\{a, 1-a\}$$

Si può facilmente intuire come N_ℓ^+/N_ℓ e N_ℓ^-/N_ℓ siano sempre compresi tra 0 e 1 in quanto rappresentano la percentuale di esempi positivi/negativi che raggiungono ℓ rispetto al totale degli esempi (sempre che raggiungono ℓ).

Esempio

Sia T l'albero di figura 6 e S il *training set* mostrato in tabella 1 (vengono mostrati solo gli esempi di S che sono indirizzati a ℓ' e ℓ''). Si deve decidere che etichette assegnare alle foglie ℓ' e ℓ'';

x_t	previsione	umidità	vento	y_t
x_1	sole	85	no	+1
x_2	sole	76	sì	−1
x_3	sole	55	sì	+1
x_4	sole	65	sì	−1
x_5	sole	82	sì	−1
x_6	sole	35	no	+1
x_7	sole	94	no	−1
x_8	sole	66	no	+1
x_9	sole	48	sì	+1

Tabella 1: Esempio di *training set*

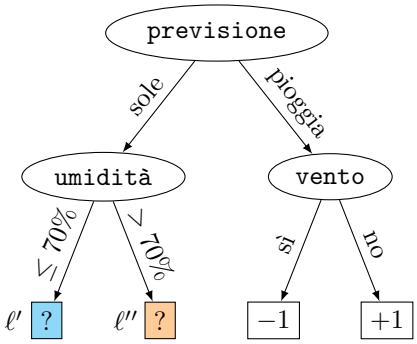

Figura 6: Esempio di *tree classifier* "in costruzione".

Si prenda ℓ':

$$S_{\ell'} = \{(x_3,+1),(x_4,-1),(x_6,+1),(x_8,+1),(x_9,+1)\} \quad N_{\ell'} = 5$$
$$S_{\ell'}^+ = \{(x_3,+1)(x_6,+1),(x_8,+1),(x_9,+1)\} \quad N_{\ell'}^+ = 4 \quad \frac{N_{\ell'}^+}{N_{\ell'}} = 0.8$$
$$S_{\ell'}^- = \{(x_4,-1)\} \quad N_{\ell'}^- = 1 \quad \frac{N_{\ell'}^-}{N_{\ell'}} = 0.2$$

L'ottanta percento degli esempi che raggiungono ℓ' ha etichetta positiva, si può quindi affermare che l'etichetta $y_{\ell'} = +1$.

Si prenda infine ℓ'':

$$S_{\ell''} = \{(x_1,+1),(x_2,-1),(x_5,-1),(x_7,-1)\} \quad N_{\ell''} = 4$$
$$S_{\ell''}^+ = \{(x_1,+1)\} \quad N_{\ell''}^+ = 1 \quad \frac{N_{\ell''}^+}{N_{\ell''}} = 0.25$$
$$S_{\ell''}^- = \{(x_2,-1),(x_5,-1),(x_7,-1)\} \quad N_{\ell''}^- = 3 \quad \frac{N_{\ell''}^-}{N_{\ell''}} = 0.75$$

Il settantacinque percento degli esempi che raggiungono ℓ'' ha etichetta negativa, si può quindi affermare che l'etichetta $y_{\ell''} = -1$.

3.2.3 Crescita dell'albero e *training error*

Si supponga di sostituire una foglia ℓ con un nodo interno e due nuove foglie ℓ' e ℓ'', come mostrato in figura 7. **Può il *training error* del nuovo albero espanso crescere rispetto a quello originale?**

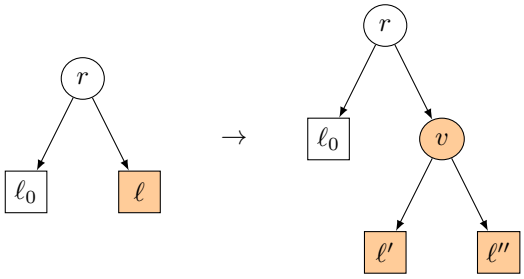

Figura 7: Un passaggio della crescita dell'albero: la foglia ℓ viene rimpiazzata da un nodo interno v con due nuove foglie ℓ' e ℓ''.

L'apporto che la foglia ℓ dà al *training error* è:

$$\psi\left(\frac{N_\ell^+}{N_\ell}\right) N_\ell \tag{1}$$

Gli esempi con etichetta positiva che vengono indirizzati a ℓ saranno ora divisi tra ℓ' e ℓ'':

$$N_\ell^+ = N_{\ell'}^+ + N_{\ell''}^+ \tag{2}$$

Si può quindi ottenere che:

$$\begin{aligned}
\frac{N_\ell^+}{N_\ell} &= \frac{N_{\ell'}^+ + N_{\ell''}^+}{N_\ell} \\
&= \frac{N_{\ell'}^+}{N_\ell} + \frac{N_{\ell''}^+}{N_\ell} \\
&= \frac{N_{\ell'}^+}{N_\ell} \cdot \frac{N_{\ell'}}{N_{\ell'}} + \frac{N_{\ell''}^+}{N_\ell} \cdot \frac{N_{\ell''}}{N_{\ell''}} \\
&= \frac{N_{\ell'}^+}{N_{\ell'}} \cdot \frac{N_{\ell'}}{N_\ell} + \frac{N_{\ell''}^+}{N_{\ell''}} \cdot \frac{N_{\ell''}}{N_\ell}
\end{aligned} \tag{3}$$

Dati (1) e (3) si può dire che l'apporto di ℓ è:

$$\psi\left(\frac{N_\ell^+}{N_\ell}\right) N_\ell = \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}} \cdot \frac{N_{\ell'}}{N_\ell} + \frac{N_{\ell''}^+}{N_{\ell''}} \cdot \frac{N_{\ell''}}{N_\ell}\right) N_\ell \tag{4}$$

Si noti che ψ è una funzione concava. Questo permette di poter applicare la disuguaglianza di Jensen (valida con $a, b \in \mathbb{R} \wedge \mu \in [0, 1]$):

$$\psi(a\mu + b(1-\mu)) \geq \mu\psi(a) + (1-\mu)\psi(b) \tag{Jensen}$$

$$\begin{aligned}
\psi\left(\frac{N_{\ell'}^+}{N_{\ell'}} \cdot \frac{N_{\ell'}}{N_\ell} + \frac{N_{\ell''}^+}{N_{\ell''}} \cdot \frac{N_{\ell''}}{N_\ell}\right) N_\ell &\geq \cancel{N_\ell} \frac{N_{\ell'}}{\cancel{N_\ell}} \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right) + \cancel{N_\ell} \frac{N_{\ell''}}{\cancel{N_\ell}} \psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right) \\
&\geq \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right) N_{\ell'} + \psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right) N_{\ell''}
\end{aligned} \tag{5}$$

Chiaramente si dovrà avere che:

$$\frac{N_{\ell''}}{N_\ell} = 1 - \frac{N_{\ell'}}{N_\ell}$$

Questo è facilmente verificabile a partire da (2). Infine, dati (4) e (5) si ha:

$$\underbrace{\psi\left(\frac{N_\ell^+}{N_\ell}\right)N_\ell}_{\text{apporto di } \ell} \geq \underbrace{\psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right)N_{\ell'}}_{\text{apporto di } \ell'} + \underbrace{\psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right)N_{\ell''}}_{\text{apporto di } \ell''}$$

Questo dimostra che, **facendo crescere l'albero, il *training error* non aumenta**.

Una foglia ℓ viene detta pura se non contribuisce ad aumentare il *training error*, ovvero:

$$N_\ell^+ \in \{0, N_\ell\}$$

In qualsiasi albero il *training error* è maggiore di zero ($\ell_S(h_T) > 0$) almeno che non abbia solo foglie pure.

3.2.4 Algoritmo generale

Verrà ora mostrato un algoritmo generale per la cotruzione di un albero binario a partire da un *training set* S:

1. **Inizializzazione:**
   (a) Crea un albero T con solo la radice ℓ
   (b) $S_\ell = S$
   (c) $y_\ell =$ etichetta più frequente in S_ℓ
2. ***Main loop*:**
   (a) Scegli una foglia ℓ e sostituiscila con un nodo interno v e due nuove foglie ℓ' e ℓ''
   (b) Scegli un attributo i e un test $f : \mathcal{X}_i \to \{1, 2\}$
   (c) Associa il test f a v e partiziona S_ℓ in due sottoinsiemi:

$$S_{\ell'} = \{(x_t, y_t) \in S_\ell : f(x_{t,i}) = 1\} \qquad \text{e} \qquad S_{\ell''} = \{(x_t, y_t) \in S_\ell : f(x_{t,i}) = 2\}$$

   (d) Associa a ℓ' l'etichetta più frequente in $S_{\ell'}$
   (e) Associa a ℓ'' l'etichetta più frequente in $S_{\ell''}$

3.3 *Overfitting*

Se il numero di nodi dell'albero è troppo alto rispetto alla cardinalità di S si potrà avere dell'*overfitting*. Per questo motivo, la scelta della foglia da espandere dovrebbe garantire approssimativamente la riduzione massima del *training error*.

Nella pratica, per calcolare il *training error* si usano funzioni diverse da $\psi(p) = \min\{p, 1-p\}$. Questo perchè ψ può essere problematica in alcuni casi. Un esempio è il seguente:

$$p = \frac{N_\ell^+}{N_\ell} = 0.8 \qquad q = \frac{N_{\ell'}^+}{N_{\ell'}} = 0.6 \qquad r = \frac{N_{\ell''}^+}{N_{\ell''}} = 1 \qquad \alpha = \frac{N_{\ell'}^+}{N_\ell} = 0.5$$

$$\underbrace{\psi(p)}_{\text{apporto di } \ell} - \overbrace{(\alpha\psi(q)}^{\text{apporto di } \ell'} + \underbrace{(1-\alpha)\psi(r)}_{\text{apporto di } \ell''}) = 0.2 - (0.5 \cdot 0.4 + 0.5 \cdot 0) = 0$$

In questo passaggio il cambiamento che si avrebbe rimpiazzando la foglia ℓ sarebbe nullo e quindi non verrebbe scelto dall'algoritmo. Potrebbe inoltre succedere che tutte le foglie diano questo risultato facendo quindi bloccare l'algoritmo.

Per correggere questo problema vengono usate altre funzioni ψ. Queste funzioni sono simili a quella già vista in quanto simmetriche attorno a $1/2$ e nulle agli estremi ($\psi(0) = \psi(1) = 0$). Alcune funzioni usate sono:

- **Funzione di Gini:** $\psi_2(p) = 2p(1-p)$
- **Entropia scalata:** $\psi_3(p) = -\frac{p}{2}\log_2(p) - \frac{1-p}{2}\log_2(1-p)$
- $\psi_4(p) = \sqrt{p(1-p)}$

Come si può vedere in figura 8, valgono le seguenti disuguaglianze ($\psi_1(p) = \min\{p, 1-p\}$):

$$\psi_1(p) \leq \psi_2(p) \leq \psi_3(p) \leq \psi_4(p)$$

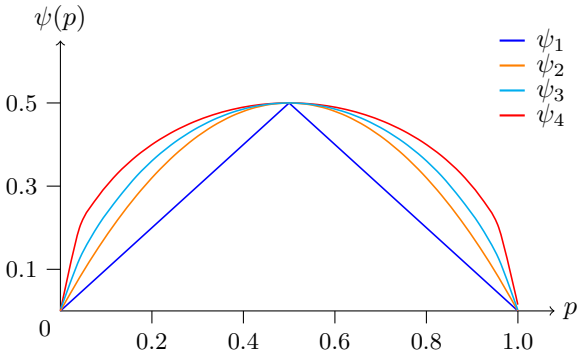

Figura 8: Grafici delle funzioni ψ

3.4 Interpretabilità

Una proprietà interessante dei *tree predictors* per la classificazione binaria è che possono essere rappresentati con una proposizione logica in forma normale disgiuntiva (DNF). Questa rappresentazione è ottenuta considerando le clausule (congiunzione di predicati) che risultano dai test che si trovano sui percorsi che portano ad un etichetta $+1$.

Un esempio è la seguente DNF ricavata dall'albero in figura 9:

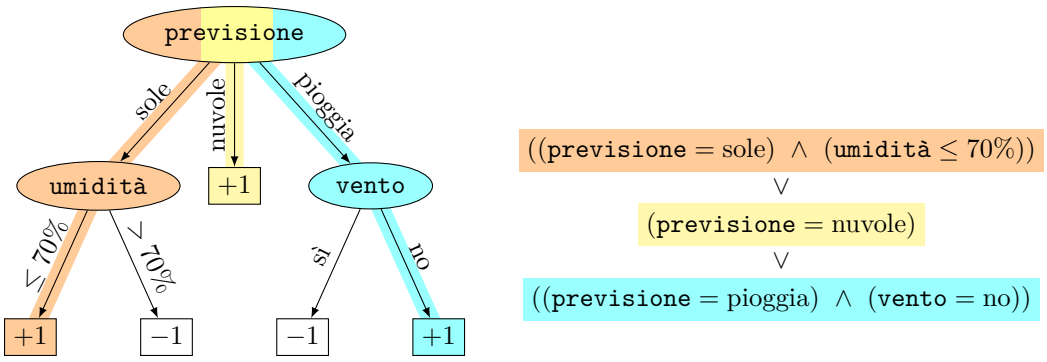

Figura 9: *Tree predictor*

Questa rappresentazione "logica" dell'albero è molto intuitiva e permette di essere manipolata attraverso le regole della logica preposizionale, permettendo quindi una possibile semplificazione del classificatore. Soprattutto, questa rappresentazione fornisce una descrizione interpretabile della conoscenza del *learning algorithm* estratta dal *training set*.

4 *Statistical Learning*