ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO di
INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE
"Guglielmo Marconi"
DEI

CORSO DI LAUREA IN
Ingegneria dell'Automazione

TESI DI LAUREA
in
Controlli Automatici T-1

MACHINE LEARNING E CONFRONTO FRA REGRESSIONE LOGISTICA E SUPPORT VECTOR MACHINE

CANDIDATO

Tancredi Bosi

RELATORE

Ch iar.mo Prof. Roberto Diversi

Anno Accademico
2022/2023

Sessione
I

Indice

Indice delle figure

Indice delle tabelle

INTRODUZIONE

Il machine learning è un campo dell'Intelligenza artificiale che si occupa dello sviluppo di algoritmi che possano apprendere e fare previsioni o prendere decisioni basandosi su dati. La storia del machine learning risale alla metà del XX secolo con lo sviluppo dei primi sistemi informatici e da allora è uno dei settori che si è sviluppato maggiormente. Gli ambiti in cui è utilizzato sono molteplici e gli algoritmi di machine learning facilitano lo svolgimento di molte attività. Sono presenti molti algoritmi, bisogna saper valutare quale sia il migliore nel caso specifico da affrontare. Infatti, non esiste un algoritmo ottimale, ognuno ha le sue caratteristiche ed è necessario capire quale sia il migliore di caso in caso rispettando le specifiche e valutando i risultati ottenuti.

Uno degli algoritmi più utilizzati è la regressione logistica. Questo modello è utilizzato per problemi di classificazione, dove l'obiettivo è predire a quale delle due o più classi appartenga una istanza in ingresso. La regressione logistica elabora i dati in modo da modellare la funzione di probabilità dell'appartenenza di un'istanza ad una determinata classe.

Un altro algoritmo largamente applicato è il *support vector machine*, ovvero un modello che cerca di trovare l'iperpiano che separi al meglio le classi. Questo algoritmo è molto efficace se utilizzato con dati aventi una grande quantità di attributi e può essere utilizzato sia con problemi di classificazione lineare, sia non lineare.

I due algoritmi sopra citati sono differenti nella loro struttura e hanno caratteristiche differenti: il primo è più semplice, interpretabile e rapido; il secondo risulta essere più preciso, più robusto, ma anche computazionalmente più pesante.

1. IL MACHINE LEARNING

Il machine learning è la programmazione di computer per ottimizzare certi criteri di performance utilizzando dati o esperienze passate. Si ha quindi un modello definito in base a certi parametri e l'apprendimento è l'esecuzione di un programma del computer per ottimizzare tali parametri attraverso i dati di allenamento o esperienze passate. Viene utilizzata quindi la teoria della statistica per creare modelli matematici predittivi, per fare predizione di eventi futuri, o descrittivi, per capire meglio i dati analizzati. Vengono utilizzate poi le basi delle scienze informatiche, infatti sono necessari algoritmi nella fase di addestramento per risolvere il problema di ottimizzazione che siano efficienti per minimizzare i tempi di elaborazione dei dati e il consumo di risorse. In certe applicazioni si ha che la efficienza dell'algoritmo (complessità spaziale e temporale) è tanto importante quanto la precisione della predizione.

Il sistema di apprendimento di un algoritmo di machine learning può essere diviso in tre parti:

1. Processo decisionale: sulla base di dati in input l'algoritmo produce una stima su un modello nei dati.
2. Funzione di errore: viene valutata una funzione di errore per la previsione del modello. Se viene utilizzata una funzione di errore nota, essa può essere utilizzata per un confronto tra algoritmi in base alla loro accuratezza.
3. Processo di ottimizzazione: l'algoritmo ripete un processo di valutazione e ottimizzazione dei pesi utilizzati in esso al fine di minimizzare la funzione di errore, e quindi raggiungendo una certa soglia di accuratezza.

1.1 Storia del Machine Learning

Essendo il machine learning un sottoinsieme dell'intelligenza artificiale, la sua storia è parallela e spesso non bene delineata rispetto all'IA. Nel 1943 Pitts e McCulloch pubblicarono il primo modello matematico di una rete neurale per creare algoritmi che imitassero il processo di pensiero caratteristico degli umani. Nel 1950 Alan Turing propose il "test di Turing": una macchina può essere definita intelligente se essa può convincere un umano che anch'essa sia umana . Sempre nello stesso anno, Minsky e Edmonds costruirono il primo computer basato su rete neurale, lo SNARC. Il termine "machine learning" fu invece coniato nel 1959 da Arthur Samuel, un impiegato di IBM che sviluppò un programma per giocare a dama nel quale veniva utilizzata la strategia "minmax" per minimizzare le possibilità di perdita del computer. Nel 1957 Rosenblatt provò a progettare la prima rete neurale

informatica, chiamata “percettrone”, costruita per ricevere input visivi, come immagini, e creare etichette e categorie come output.

Nel 1963 Michie creò un programma chiamato “Menace” che aveva la capacità di apprendere come giocare una partita perfetta di Tris (*Tic-tac-toe*). Quattro anni dopo viene definito l’algoritmo *nearest neighbor*, il quale diede la possibilità ai computer di riconoscere pattern. Nel 1979 un gruppo di ricercatori dell’Università di Stanford creò un robot (“The Cart”) capace di evitare ostacoli all’interno di una stanza. Sejnowski elaborò un programma chiamato NetTalk nel 1985 in grado di imparare a pronunciare le parole in modo simile a come le impara un bambino. Nel 1995 vennero pubblicati due articoli molto importanti: il primo sull’algoritmo *random decision forest* da Tin Kam Ho e il secondo sull’algoritmo *support vector machine* da Vapnik e Cortes. Nel 1996 venne sviluppato il programma Deep Blue di IBM in grado di giocare a scacchi e battere il campione del mondo Garry Kasparov provocando così un grande interesse per le scienze informatiche.

Nel XXI secolo il mondo del machine learning si è sviluppato esponenzialmente. Nel 2002 Torch rilasciò la prima libreria software pubblica per il machine learning. Nel 2006 fu coniato il termine “Deep learning” per descrivere gli algoritmi che aiutano i computer a riconoscere oggetti differenti e caratteri di testo in immagini e video. Nel 2010 Microsoft realizzò un dispositivo per console in grado di tracciare 20 movimenti umani, 30 volte al secondo. Un anno dopo, IBM fece gareggiare il sistema Wattson in un programma televisivo statunitense di domande di cultura generale a cui i concorrenti erano tenuti a rispondere nel minor tempo possibile, Wattson fu in grado di vincere ripetutamente nel programma. Molto virale fu anche la rete neurale Google Brain sviluppata da Google in grado di imparare autonomamente a riconoscere gatti nei video di YouTube. Nel 2014 Facebook pubblicò un algoritmo di DeepFace in grado di identificare individui nelle fotografie.

L’avanzamento tecnologico del machine learning è molto rapido e in costante evoluzione. Con l’aumento dell’uso della tecnologia nella quotidianità l’argomento è sempre più virale e di interesse, un esempio è l’avvento di ChatGPT di OpenAI nel 2022. La quantità di dati prodotti al giorno d’oggi è inquantificabile e un’analisi può solamente essere fatta da un sistema informatico. Infatti, per molti problemi è più sensato occuparsi dei dati prima di pensare quale algoritmo applicare. Un buon approccio consiste perciò nel conoscere la tipologia di dati che si sta trattando e visualizzarne la distribuzione generale, se possibile, e scegliere poi l’algoritmo secondo cui elaborare tali dati.

1.2 Applicazioni del Machine Learning

Le applicazioni del machine learning sono molteplici, ed aumentano molto rapidamente, fino ad essere parte della quotidianità. Questo accade dato che esso è riuscito a rivoluzionare molti settori, come quello medico, finanziario, ingegneristico e del marketing, grazie alla capacità di analizzare grandi quantità di dati e fornire previsioni accurate.

Per quanto riguarda l'ambito medico, gli algoritmi di machine learning sono utilizzati per la diagnosi precoce di malattie. Si prendono in analisi dati relativi a pazienti con o senza la malattia considerata, e, a fronte di nuovi pazienti, si è in grado di diagnosticare o meno tale problema con una certa accuratezza. È uno strumento molto utile in quanto apporta un aiuto ai medici nell'identificazione di problemi e spesso può prevenire aggravamenti.

In ambito finanziario vengono sviluppati algoritmi per la previsione del mercato azionario, il rilevamento delle frodi e le valutazioni dei rischi in determinati investimenti o prestiti di credito. È molto importante anche la parte di personalizzazione dell'esperienza del cliente migliorata grazie al machine learning, infatti sono spesso presenti *chat bot* in grado di prestare servizio all'utente aiutandolo con la gestione del proprio portafoglio bancario.

Nel marketing il machine learning offre contributi enormi grazie alla sua capacità di analizzare grandi quantità di dati. Si riescono per esempio ad elaborare gli interessi degli utenti online per offrire loro ciò di cui hanno bisogno, pubblicizzando i prodotti coinvolti ed ottimizzando quindi la campagna pubblicitaria. Vengono realizzate inoltre previsioni delle vendite, valutazioni sui guadagni, segmentazione del pubblico, servizi clienti più rapidi e decisioni sui prezzi da attribuire a certi prodotti, come quello di una macchina.

Un caso molto noto di applicazione del machine learning è quello del riconoscimento e classificazione di immagini. Possono essere infatti identificati oggetti, persone, animali e caratteri, portando così ad un suo utilizzo in vari campi, come quello del riconoscimento dei volti. In ambito ingegneristico e architettonico vengono ottimizzate le topologie delle strutture e dei centri urbani, si addestrano veicoli per una guida autonoma e si creano modelli predittivi nell'aeronautica.

Le applicazioni degli algoritmi di machine learning sono perciò presenti in ogni ambito immaginabile, i sopra citati sono solamente alcuni dei più noti.

1.3 Approcci del Machine Learning

Gli approcci di apprendimento del machine learning sono tradizionalmente divisi in tre categorie: supervisionato, non supervisionato e per rinforzo.

L'apprendimento supervisionato mira ad istruire un sistema informatico in modo da consentirgli di elaborare automaticamente previsioni sui valori di uscita di un sistema rispetto ad un dataset di ingresso sulla base di coppie di input e output forniti inizialmente. Si produce quindi una funzione in grado di apprendere dai risultati forniti durante la fase di allenamento e produrre output il più possibile corretti per tutti gli input non utilizzati in allenamento, set di verifica (o test).

Nell'apprendimento non supervisionato non si hanno invece i risultati del dataset fornito in input, ma solo dati. L'obiettivo è perciò riconoscere certe strutture o *pattern* che occorrono più frequentemente di altre e cercare di effettuare previsioni su input successivi in base a tali caratteristiche identificate.

L'apprendimento per rinforzo si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. È in grado di scegliere le azioni da compiere per il raggiungimento di obiettivi. Non è importante quindi la singola azione, ma la sequenza di corrette azioni per conseguire l'obiettivo. La qualità della decisione è data da un valore di ricompensa che ha lo scopo di incoraggiare i comportamenti corretti dell'algoritmo.

1.4 Output di un sistema di Machine Learning

Dato l'output di un sistema di machine learning, si possono individuare alcune categorie: classificazione, regressione, *clustering* e riduzione della dimensionalità.

Nella classificazione l'output è discreto, infatti, i risultati sono suddivisi in due o più classi e il sistema di apprendimento deve produrre un modello che categorizzi gli input non ancora analizzati a una o più di tali classi. Nella regressione, invece, l'output e il modello utilizzato sono continui. Un caso particolare della regressione è il *ranking*, ovvero l'ordinamento in base ad una certa scala delle istanze in ingresso.

Se si è nel caso di apprendimento non supervisionato, si parla di *clustering*: le classi non sono note a priori e l'input deve essere diviso in gruppi. Le tecniche di *clustering* si basano su misure della somiglianza tra i dati in input che vengono rappresentate come distanze in una certa dimensione. Un'altra tecnica dell'apprendimento non supervisionato è quella della riduzione della dimensionalità.

Essa consiste nell'eliminare i rumori dei dati e nel combinare le informazioni correlate e ridondanti, riducendo un dataset di dimensione $\mathbb{R}^n$ in uno spazio di dimensione $\mathbb{R}^k$ con $k < n$.

Quindi, sia classificazione che regressione sono problemi dell'apprendimento supervisionato dove c'è un input X e un output Y, e l'obiettivo è apprendere la mappatura fra input e output. L'approccio del machine learning è assumere un modello definito rispetto ad un set di parametri $y = g(x \mid \theta)$, dove $g(\cdot)$ è il modello e θ il suo set di parametri. Nella regressione Y è un numero reale, nella classificazione un numero intero che identifica una classe. $g(\cdot)$ è quindi la funzione di regressione o, nella classificazione, la funzione discriminante che separa le istanze di classi differenti. Un programma di machine learning ottimizza il set di parametri tale da minimizzare la funzione di errore.

1.5 Selezione di un modello

Lo scopo del machine learning è fare predizioni su nuovi dati piuttosto che replicare i dati utilizzati in fase di allenamento, ovvero si vuole generare l'output corretto per un'istanza di input al di fuori del set di allenamento. La qualità di predizione su nuove istanze di un modello è detta generalizzazione. Per avere la migliore generalizzazione possibile, bisogna effettuare la selezione del modello in modo da avere la complessità della funzione di distribuzione dei dati pari alla complessità dell'ipotesi di distribuzione delle classi H.

Scegliere la corretta complessità è fondamentale per evitare *underfitting* e *overfitting* dei dati. Si ha *underfitting* quando la complessità della classe di ipotesi è minore della reale funzione di andamento dei dati e quindi si ha un *bias* elevato. Se si prova ad utilizzare una retta come ipotesi di classe su dati con distribuzione superiore al terzo ordine, si cade in questo problema. Si ha invece *overfitting* quando la complessità dell'ipotesi di distribuzione delle classi è più complessa del dovuto, avendo quindi un'alta varianza. Ne consegue una identificazione molto precisa delle istanze di allenamento, ma una bassa robustezza al rumore. In questo secondo caso, avere più dati di allenamento aiuta, ma fino ad un certo punto.

Per prevenire i problemi di *overfitting* è possibile utilizzare tecniche di regolarizzazione. Regolarizzare un modello significa cambiarne il comportamento di apprendimento durante la fase di addestramento. Si previene l'*overfitting* aggiungendo una penalità sulla complessità del modello. Si modifica quindi la funzione di perdita che l'algoritmo cerca di ottimizzare; le regolarizzazioni più utilizzate sono quelle di tipo L1 e L2. Nella regolarizzazione L1 (Lasso) si aggiunge una penalità sulla somma dei valori assoluti dei pesi del modello, ovvero, si azzerano i pesi che non contribuiscono molto al modello. L1 funziona

perciò bene nel caso di selezione di attributi quando ce ne sono molti. Nella regolarizzazione L2 (Ridge) si aggiunge una penalità sulla somma dei quadrati dei pesi del modello, portando quindi ad avere pesi più distribuiti, dato che i più elevati vengono ridotti. L2 è quindi utile quando si hanno molte variabili altamente correlate. Di seguito è mostrato cosa accade effettuando una regolarizzazione, il modello (linea rossa) ha problemi di *overfitting* essendo creato attraverso un polinomio di grado 15; aggiungendo un termine di penalizzazione si può ottenere la linea verde che ha una robustezza al rumore molto migliore.

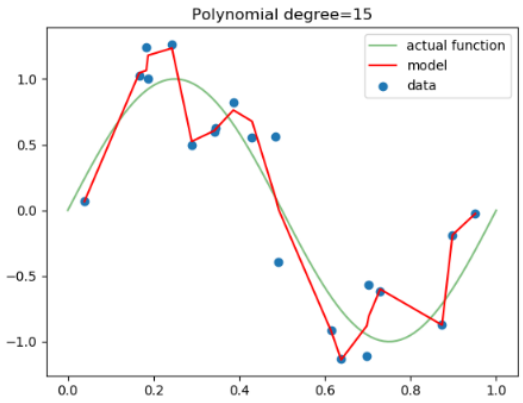


Figura 1: Regolarizzazione di un modello.

Per la selezione del modello, si effettua quindi un triplo *trade-off* tra la complessità dell'ipotesi scelta, la quantità di dati nel set di allenamento e l'errore di generalizzazione su nuove istanze. Aumentando il numero di dati nel set di allenamento, diminuisce l'errore di generalizzazione; aumentando la complessità del modello, l'errore di generalizzazione decresce inizialmente per poi aumentare.

Si può misurare la generalizzazione di un modello avendo accesso a dati al di fuori di quelli di allenamento. Per fare ciò si divide il dataset in due parti: un set di allenamento e un set di validazione utilizzato per verificare la generalizzazione.

Si parte quindi da un dataset $X = \left\{\mathbf{x}^t, \mathbf{r}^t\right\}_{t=1}^{N}$ di N istanze, dove x^t è un input arbitrario di dimensione d e r^t è l'output desiderato associato all'input. r^t è 0/1 per il caso di due classi, un vettore binario di dimensione K per la classificazione di più di due classi, e un valore reale per la regressione.

L'obiettivo è costruire una buona approssimazione di r^t utilizzando il modello $g(x^t \mid \theta)$, e per fare ciò sono necessarie tre scelte. In primo luogo, va selezionato il modello $g(x \mid \theta)$ da utilizzare per l'apprendimento; $g(\cdot)$ definisce l'ipotesi di distribuzione delle classi. Successivamente, bisogna definire la funzione di perdita $L(\cdot)$ per verificare la differenza tra l'output desiderato r^t e

l'approssimazione $g(x^t \mid \theta)$: l'errore di approssimazione (o perdita) è la somma delle funzioni di perdita calcolate su tutte le istanze individualmente

$$E(\theta \mid X) = \sum_t L(r^t, g(x^t \mid \theta))$$

Infine, è necessaria una procedura di ottimizzazione per trovare $\bar{\theta}$ tale che venga minimizzato l'errore totale:

$$\bar{\theta} = \arg\min_{\theta} E(\theta \mid X)$$

Dove $\arg\min$ fornisce come risultato l'argomento che minimizza la funzione.

Gli algoritmi di machine learning si diversificano per il modello di ipotesi utilizzato, per la misura della perdita impiegata e per la procedura di ottimizzazione utilizzata.

1.6 Valutazione dei risultati ottenuti

L'errore ottenuto non è l'unico criterio da tenere in considerazione nel confronto tra algoritmi di apprendimento. Infatti, in base all'applicazione specifica può essere rilevante considerare anche la funzione di rischio, il tempo e la complessità dell'allenamento e della verifica, l'interpretabilità e la facilità di implementazione.

Nel caso di due classi, può essere identificata una matrice che copra tutte le possibili combinazioni dei risultati (*confusion matrix*):

	Classe predetta		
Classe effettiva	Positiva	Negativa	Totale
Positiva	tp: *true positive*	tn: *true negative*	p
Negativa	fp: *false positive*	fn: *false negative*	n
Totale	p'	n'	N

Tabella 1: matrice delle combinazioni dei risultati.

Per la classificazione, in particolare per i problemi a due classi, vengono definite alcune misure particolari identificate dalle seguenti formule:

Error	(fp + fn) / N
Accuracy	(tp + tn) / N = 1- error

tp-rate	tp / p
fp-rate	fp / n
Precision	tp / p’
Recall	tp / p = tp-rate
Sensitivity	tp / p = tp-rate
Specificity	tn / n = 1 – fp-rate

Tabella 2: misure delle performance.

Per valutare un algoritmo sono spesso utilizzati i valori di *accuracy*, ovvero la quantità di predizioni corrette diviso il numero totale di predizioni, di *precision*, il valore di istanze positive classificate correttamente diviso il numero delle predizioni positive totali, e di *recall*, il valore di istanze positive classificate correttamente diviso il numero di istanze effettivamente positive.

Un altro fattore molto importante calcolato direttamente a partire dai valori di *precision* e *recall*, è *f1-score*, ovvero un risultato spesso utilizzato per confrontare modelli su problemi con due classi. Esso viene calcolato come la media armonica tra i due valori sopra detti:

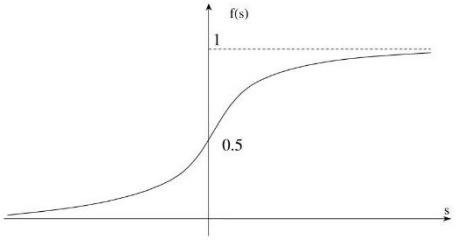
$$\text{f1-score} = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Nel caso di più classi ($K > 2$), la *confusion matrix* è una matrice $K \times K$ dove l’elemento (i, j) contiene il numero di istanze che appartengono alla classe C_i, ma sono assegnate alla classe C_j.

2. REGRESSIONE LOGISTICA

La regressione logistica è un algoritmo di analisi dei dati supervisionato che decreta il modello di classificazione direttamente dal discriminante. Per questo algoritmo si prende come stimatore della probabilità di una classe $P(C_i \mid \mathbf{x})$ la funzione sigmoidale:

$$y = \hat{P}(C_i \mid \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$



Per applicare la regressione logistica ci sono alcune assunzioni senza le quali l'algoritmo avrebbe pessime prestazioni:

- Non deve esserci una correlazione tra le variabili indipendenti (mancanza di multicollinearità).
- Le osservazioni devono essere indipendenti.
- Ci deve essere una relazione lineare tra la funzione logistica del risultato e ognuna delle variabili indipendenti continue.
- Non devono essere presenti eccezioni (*outliers*) che influenzino in modo considerevole i dati.

Vengono testati diversi valori di $\mathbf{w}$, mediante iterazioni, per eseguire un'ottimizzazione al fine di migliorare la qualità della predizione tramite probabilità logaritmica. Queste iterazioni producono la funzione di verosimiglianza logaritmica; l'algoritmo cerca di massimizzare tale funzione, o equivalentemente minimizzarne l'errore, per trovare la migliore stima dei parametri.

2.1 Discriminazione parametrica

Considerando il caso in cui ci siano due classi da discriminare, le probabilità vengono definite come $y \equiv P(C_1 \mid \mathbf{x})$ e $P(C_2 \mid \mathbf{x}) = 1 - y$. Quindi, nella classificazione si sceglie C_1 se $y > 0.5$, o equivalentemente se $\frac{y}{1-y} > 1$ o $\log\left(\frac{y}{1-y}\right) > 0$, e C_2 altrimenti, dove $\log\left(\frac{y}{1-y}\right)$ è detta trasformazione logistica (*logit*). Nel caso di due classi con la matrice di covarianza in comune e distribuzione gaussiana, la trasformazione logistica è lineare. Infatti,

$$\text{logit}\left(P(C_1 \mid \mathbf{x})\right) = \log \frac{P(C_1 \mid \mathbf{x})}{1 - P(C_1 \mid \mathbf{x})} = \log \frac{P(C_1 \mid \mathbf{x})}{P(C_2 \mid \mathbf{x})} = \log \frac{P(\mathbf{x} \mid C_1)}{P(\mathbf{x} \mid C_2)} + \log \frac{P(C_1)}{P(C_2)}$$

$$= \log \frac{(2\pi)^{-d/2} \mid \Sigma \mid^{-1/2} e^{-\left(\frac{1}{2}(\mathbf{x}-\mu_1)^T \Sigma^{-1}(\mathbf{x}-\mu_1)\right)}}{(2\pi)^{-d/2} \mid \Sigma \mid^{-1/2} e^{-\left(\frac{1}{2}(\mathbf{x}-\mu_2)^T \Sigma^{-1}(\mathbf{x}-\mu_2)\right)}} + \log \frac{P(C_1)}{P(C_2)} = \mathbf{w}^T \mathbf{x} + b$$

dove

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$b = -\frac{1}{2}(\mu_1 + \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) + \log \frac{P(C_1)}{P(C_2)}$$

È da notare come le espressioni ricavate per $\mathbf{w}$ e b valgono solamente se le due classi hanno distribuzione gaussiana; si denomina questo caso particolare *linear discriminant analysis*. La regressione logistica non richiede invece alcuna ipotesi sulla distribuzione delle classi.

Ricavando $P(C_1 \mid \mathbf{x})$ si ottiene la funzione logistica, anche detta funzione sigmoidale:

$$P(C_1 \mid \mathbf{x}) = sigmoid(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$

Quindi, durante la fase di allenamento, si stimano i valori medi e la matrice di covarianza per calcolare i parametri del discriminante. Durante la fase di test, dato $\mathbf{x}$, viene calcolato $y = sigmoid(\mathbf{w}^T \mathbf{x} + b)$ e si sceglie C_1 se $y > 0.5$ dato che $sigmoid(0) = 0.5$.

2.2 Metodo del gradiente in discesa

Negli algoritmi con approccio basato sul discriminante come la regressione logistica, i parametri da ottimizzare sono quelli del discriminante, e l'ottimizzazione viene effettuata per minimizzare l'errore di classificazione di un set di allenamento. Prendendo un set di parametri $\mathbf{w}$ e l'errore $E(\mathbf{w} \mid X)$ dei parametri sul set di allenamento dato X, si cerca $\bar{\mathbf{w}} = \arg\min_{\mathbf{w}} E(\mathbf{w} \mid X)$. Spesso si deve ricorrere ad un metodo di ottimizzazione iterativo, come il metodo del gradiente in discesa (*gradient descent*).

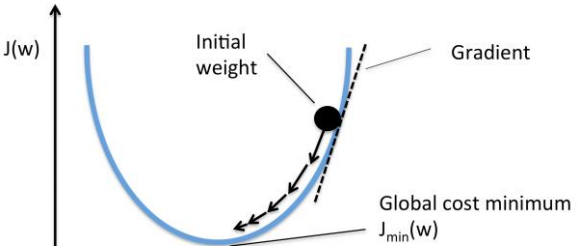


Figura 2: Metodo del gradiente in discesa

Il gradiente dell'errore è composto dalle sue derivate parziali $\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_d} \right]^T$ e la procedura del gradiente in discesa per minimizzare E parte con $\mathbf{w}$ casuale e, ad ogni step, aggiorna $\mathbf{w}$ nella direzione opposta a quella del gradiente:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} \quad \forall i$$
$$w_i = w_i + \Delta w_i$$

Dove α è il fattore di apprendimento, e determina quanto muoversi nella direzione scelta. Quando si raggiunge un minimo, la derivata vale 0 e la procedura termina. Questo significa che il minimo trovato potrebbe essere locale e non assoluto per le funzioni con più di un minimo. È fondamentale la scelta del fattore α: se scelto troppo piccolo si raggiunge la convergenza troppo lentamente, se troppo grande ci possono essere oscillazioni o divergenza.

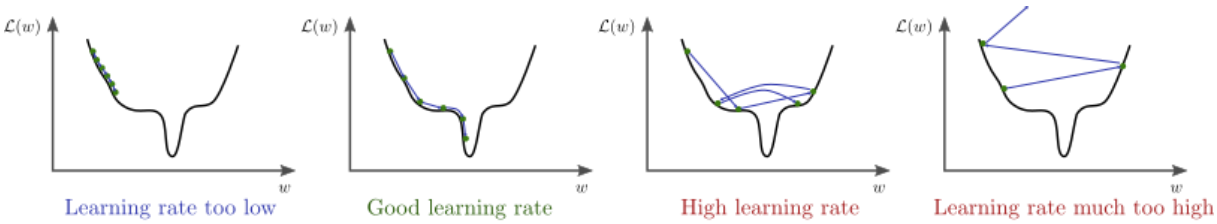


Figura 3: Possibili scenari con scelta del fattore di apprendimento

2.3 Discriminazione logistica

Prendendo per esempio due classi, $X = \{\mathbf{x}^t, r^t\}$, dove $r^t = 1$ se $x^t \in C_1$ e $r^t = 0$ se $x^t \in C_2$. Se si assume che, dato $\mathbf{x}^t$, $\mathbf{r}^t$ abbia distribuzione bernoulliana, si ha allora la funzione di verosimiglianza:

$$l(\mathbf{w}, b \mid X) = \prod_t \left(y^t\right)^{(r^t)} \left(1 - y^t\right)^{(1-r^t)}$$

Dovendo massimizzare la funzione di verosimiglianza, si può equivalentemente minimizzare una funzione di errore definita come $E = -\log(l)$, e in questo caso si ha l'entropia incrociata (*cross-entropy*):

$$E(\mathbf{w},b \mid X) = -\sum_t r^t \log(y^t) + (1-r^t)\log(1-y^t)$$

Avendo y non lineare (funzione sigmoidale), si usa il metodo del gradiente in discesa sopra descritto per minimizzare l'entropia incrociata.

$$\Delta w_j = -\alpha \frac{\partial E}{\partial w_j} = \alpha \sum_t \left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t} \right) y^t (1-y^t) x_j^t = \alpha \sum_t (r^t - y^t) x_j^t \quad j = 1,\ldots,d$$

$$\Delta b = -\alpha \frac{\partial E}{\partial b} = \alpha \sum_t (r^t - y^t)$$

I parametri w_j devono essere inizializzati, e la scelta migliore è farlo con valori randomici vicini allo 0, generalmente si prendono dall'intervallo $[-0.01, +0.01]$. Vengono presi piccoli w_j perché, se si inizializzassero con valori più grandi, si rischierebbe di avere una somma pesata che saturi la funzione sigmoidale.

Una volta terminato l'allenamento e ottenuti $\mathbf{w}$ e b finali, durante la fase di test viene calcolato $y^t = sigmoid(\mathbf{w}^T \mathbf{x}^t + b)$ e si sceglie C_1 se $y^t > 0.5$ e C_2 altrimenti. Da ciò segue che per minimizzare il numero di classificazioni errate, è necessario continuare l'apprendimento fino a che tale quantità non decresce più (è 0 se le classi sono linearmente separabili).

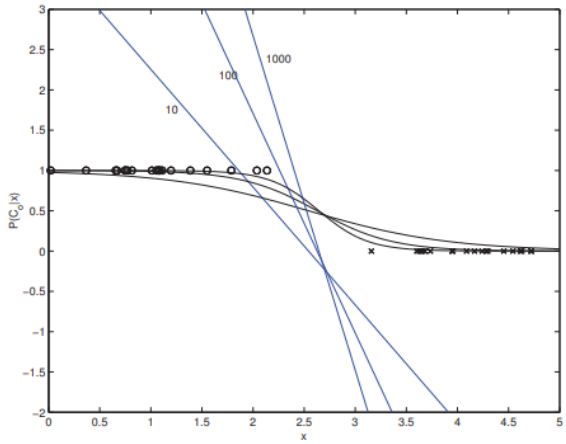


*Figura 4: evoluzione di **wx**+b e della sigmoide dopo 10, 100 e 1000 iterazioni*

Di seguito viene mostrato uno pseudocodice per una possibile implementazione della discriminazione logistica utilizzando il metodo del gradiente in discesa con due classi. Si assume che $b = 1 \ \forall t$.

```
For j = 0,...,d:
        w_j = rand(-0.01,0.01)

Ripetere fino alla convergenza:
        For j = 0,...,d:
                Δw_j ← 0
        For t = 1,...,N:
                z ← 0
                For j = 0,...,d:
                        z ← z + w_j x_j^t
                y ← sigmoid(z)
                For j = 0,...,d:
                        Δw_j ← Δw_j + (r^t − y)x_j^t
        For j = 0,...,d:
                W_j ← W_j + αΔw_j
```

2.4 Caso applicativo

Un caso applicativo possibile dell'algoritmo di regressione logistica è la classificazione di cellule in benigne e maligne. Di seguito viene affrontato il caso di diagnosi di un tumore, dataset di UCI (University of California, Irvine). Essendo la regressione logistica un algoritmo supervisionato, i casi dei pazienti presi in analisi sono già stati diagnosticati, l'output è noto. L'obiettivo dell'algoritmo è quello di riuscire a predire la diagnosi di un nuovo caso conoscendo gli attributi dati in ingresso. Le categorie dei casi prese in ingresso sono: codice identificativo del paziente, larghezza del tumore, uniformità della grandezza delle cellule, uniformità della forma delle cellule, adesione al margine, grandezza della singola cellula epiteliale, nuclei essenziali, cromatina, nucleoli normali e mitosi. Le righe del dataset, ovvero il numero di pazienti presi in analisi, sono 700, compresa la riga con le etichette delle varie colonne. Di seguito è mostrata la parte iniziale del dataset.

ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2

1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1	1	2
1017122	8	10	10	8	7	10	9	7	1	4

Tabella 3: prime istanze del dataset di UCI preso in analisi.

Per implementare l'algoritmo di predizione si è scelto il linguaggio Python. In primo luogo, è necessario importare le librerie che verranno poi utilizzate e leggere il file .csv contenente il dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


cell_df = pd.read_csv('cell_samples.csv')
cell_df.tail()
cell_df.shape
cell_df.size
cell_df.count()
cell_df['Class'].value_counts()
```

Si creano poi due variabili contenenti i casi benigni e maligni. Inoltre, si può graficare il dataset rispetto a due attributi, in modo visualizzare una distribuzione generale dei dati in ingresso. In questo caso specifico i due attributi più rilevanti sono la larghezza del tumore e l'uniformità della grandezza delle cellule.

```
#Distribution of the classes
benign_df = cell_df[cell_df['Class']==2][0:200]
malignant_df = cell_df[cell_df['Class']==4][0:200]

axes = benign_df.plot(kind='scatter', x='Clump', y='UnifSize', color='blue', label='Benign')
malignant_df.plot(kind='scatter', x='Clump', y='UnifSize', color='red', label='Malignant', ax=axes)

plt.show(block=True)
```

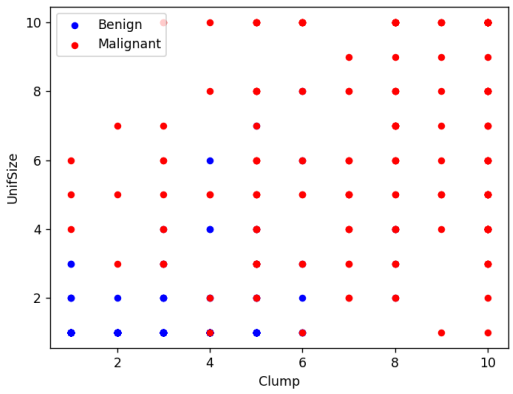


Figura 5: distribuzione delle classi secondo due attributi rilevanti.

Serve poi rimuovere le righe e le colonne non utili. La prima riga risulta superflua in quanto contiene solamente i nominativi degli attributi. Per quanto riguarda le colonne, la prima risulta inutile ai fini dell'analisi dato che contiene i codici identificativi dei pazienti; l'ultima colonna è quella riguardante l'output, quindi deve essere eliminata dal set degli attributi.

```
#Identifying unwanted rows

cell_df.dtypes

cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors ='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes


#Remove unwanted rows
cell_df.columns

feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize',
                      'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
```

Vengono poi divisi gli attributi dalla colonna di output, ovvero quella delle diagnosi:

```
# Independent variable
X = np.asarray(feature_df)

# dependent variable
y = np.asarray(cell_df['Class'])
```

Il dataset va diviso in un set di allenamento e in uno di test, usando come set di verifica il 20% del dataset di partenza. Si ottengono così quattro vettori:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

# 546 x 9
X_train.shape

# 546 x 1
y_train.shape

# 137 x 9
X_test.shape

# 137 x 1
y_test.shape
```

Serve poi allenare il modello, ottenendo un classificatore in grado di predire la classe delle istanze presenti nel set di verifica.

```
#Modeling (LogisticRegression with Scikit-learn)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

y_predict = classifier.predict(X_test)

#Evaluation (results)

print(classification_report(y_test, y_predict))
```

I risultati ottenuti possono essere visualizzati nel seguente report:

	Precision	Recall	F1-score	Support
Benigno	1.00	0.96	0.98	90
Maligno	0.92	1.00	0.96	47
Accuracy			0.97	137

Tabella 4: Risultati ottenuti con un classificatore di regressione logistica

3. SUPPORT VECTOR MACHINE

Il Support Vector Machine è un algoritmo del machine learning supervisionato che mappa le istanze di allenamento al fine di massimizzare la distanza tra le categorie. Nel caso di due classi, le nuove istanze sono mappate in questo spazio e sono predette come appartenenti ad una delle due classi in base al lato del discriminante in cui giacciono. Nella classificazione di dati in cui ogni dato appartiene ad una delle due classi, l'obiettivo è predire a quale classe appartiene un nuovo dato. Nel caso del Support Vector Machine, un dato è visto come un vettore di dimensione d, e si vuole trovare se si possono separare i dati con un iperpiano di dimensione $d-1$. Possono esserci infiniti iperpiani che classifichino i dati, è bene scegliere quello che abbia la separazione, anche detta margine, più grande tra le due classi.

Quindi, il Support Vector Machine costruisce un iperpiano in uno spazio di dimensione maggiore a quella di ingresso, che può essere usato per la classificazione e la regressione, separando le classi con un margine che sia il più elevato possibile, minimizzando quindi l'errore del classificatore.

Dato che capita spesso che il dataset da discriminare non sia linearmente separabile nello spazio di origine, questo spazio viene mappato in uno di dimensione maggiore, permettendo una separazione facilitata. Per ragioni di complessità computazionale, la mappatura effettuata negli algoritmi di Support Vector Machine garantisce che il prodotto scalare tra una coppia di istanze di input sia facilmente computabile grazie alla loro definizione in termini di funzioni di kernel $K(x,y)$ selezionate appositamente per il problema.

3.1 Iperpiani di separazione ottimale

Si parte con due classi etichettate con $-1/+1$. Prendendo come campione $\mathrm{X}=\left\{\mathbf{x}^t,r^t\right\}$, dove $r^t=+1$ se $\mathbf{x}^t\in C_1$ e $r^t=-1$ se $\mathbf{x}^t\in C_2$. L'obiettivo è trovare $\mathbf{w}$ e b tali che:

$\mathbf{w}^T\mathbf{x}^t+b\geq+1 \qquad$ per $\qquad r^t=+1$

$\mathbf{w}^T\mathbf{x}^t+b\leq-1 \qquad$ per $\qquad r^t=-1$

Ovvero,

$r^t(\mathbf{w}^T\mathbf{x}^t+b)\geq+1$

Perciò, non solo si vuole che le istanze siano nella parte corretta dell'iperpiano, ma che ci sia anche una certa distanza da esso. La distanza tra l'iperpiano e l'istanza più vicina viene chiamato margine; l'obiettivo è massimizzarlo. Infatti, l'iperpiano di separazione ottimale è quello che massimizza il margine.

La distanza tra $\mathbf{x}^t$ e il discriminante è:

$$\frac{|\mathbf{w}^T\mathbf{x}^t + b|}{||\mathbf{w}||}$$

La quale, avendo $\mathbf{r}^t \in \{-1, +1\}$, può essere riscritta come:

$$\frac{\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b)}{||\mathbf{w}||}$$

Si vuole che ciò sia maggiore di un valore ρ:

$$\frac{\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b)}{||\mathbf{w}||} \geq \rho, \ \forall t$$

Di conseguenza l'obiettivo è massimizzare ρ, ma ci sono infinite soluzioni ottenibili variando $\mathbf{w}$. Perciò si fissa $\rho||\mathbf{w}||=1$ in modo tale da minimizzare $||\mathbf{w}||$ al fine di massimizzare il margine. Quindi si vuole:

$$\min \frac{1}{2}||\mathbf{w}||^2 \quad \text{tale che} \quad \mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b) \geq +1, \ \forall t$$

Da entrambe le parti dell'iperpiano ci saranno istanze che sono distanti $\frac{1}{||\mathbf{w}||}$ dall'iperpiano, avendo così un margine totale di $\frac{2}{||\mathbf{w}||}$.

Di conseguenza si ha che la complessità di questo problema di ottimizzazione dipende da d, ovvero dal numero degli input; per cercare l'iperpiano ottimale, si converte il problema in modo da avere una complessità dipendente da N, numero delle istanze di allenamento. Per fare ciò l'equazione di minimizzazione viene riscritta usando i moltiplicatori di Lagrange α^t:

$$L_p = \frac{1}{2}||\mathbf{w}||^2 - \sum_{t=1}^{N}\alpha^t[r^t(\mathbf{w}^T\mathbf{x}^t + b) - 1] = \frac{1}{2}||\mathbf{w}||^2 - \sum_{t=1}^{N}\alpha^t r^t(\mathbf{w}^T\mathbf{x}^t + b) + \sum_{t=1}^{N}\alpha^t$$

Bisogna minimizzare L_p rispetto a $\mathbf{w}$ e b, e massimizzarlo rispetto a $\alpha^t \geq 0$.

Si può equivalentemente risolvere il problema duale, ovvero massimizzare L_p rispetto a α^t rispettando alcuni limiti: il gradiente di L_p rispetto a $\mathbf{w}$ e b deve essere 0, e $\alpha^t \geq 0$:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_t \alpha^t r^t = 0$$

Si ottiene così l'espressione duale:

$$L_d = \frac{1}{2}(\mathbf{w}^T\mathbf{w}) - \mathbf{w}^T \sum_t \alpha^t r^t \mathbf{x}^t - b\sum_t \alpha^t r^t + \sum_t \alpha^t = -\frac{1}{2}(\mathbf{w}^T\mathbf{w}) + \sum_t \alpha^t = -\frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s + \sum_t \alpha^t$$

la quale va massimizzata rispetto solamente ad α^t, sotto i limiti:

$$\sum_t \alpha^t r^t = 0, \text{ e } \alpha^t \geq 0, \ \forall t$$

Una volta risolto per α^t, visto che ci sono N istanze, molte sono cancellate con $\alpha^t = 0$ e solo una piccola percentuale ha $\alpha^t > 0$. L'insieme di $\mathbf{x}^t$ tali che $\alpha^t > 0$ sono i *support vectors*. Inoltre, riprendendo che $\mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t$, $\mathbf{w}$ è una somma pesata delle istanze di allenamento selezionate come support vectors. Quindi i support vectors sono gli $\mathbf{x}^t$ che soddisfano:

$$\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b) = 1$$

e sono perciò sul limite del margine. Si può così calcolare b da qualunque support vector:

$$b = r^t - \mathbf{w}^T\mathbf{x}^t$$

Per avere stabilità numerica, b viene calcolato per ogni support vector e si prende il valor medio. Il discriminante così trovato viene chiamato s*upport vector machine* (SVM).

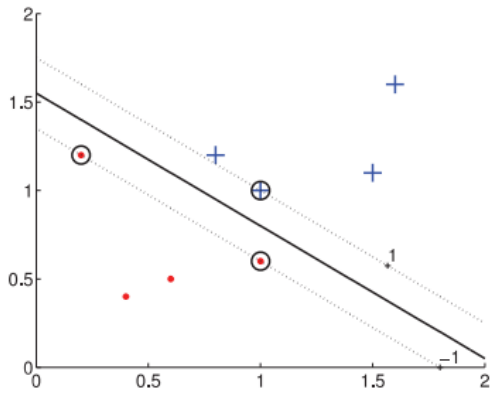


Figura 6: La linea continua è il discriminante e le linee tratteggiate definiscono il margine.

La maggior parte degli α^t sono 0, per i quali vale $\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b) > 1$, ovvero si tratta degli $\mathbf{x}^t$ che stanno sufficientemente lontano dal discriminante, i quali non hanno effetto sull'iperpiano. Le istanze che non sono support vector non portano informazione. Quindi il dataset di istanze può essere filtrato prima di utilizzare il SVM, per esempio usando un algoritmo di *k-nearest neighbors*, in modo semplificare l'ottimizzazione del SVM e ridurne la complessità.

3.2 Casi non separabili

Nel caso in cui le due classi del dataset non siano linearmente separabili, non esiste un iperpiano che le separi, quindi si cerca quello che dia il minimo errore. Si definiscono perciò delle variabili che deviino dal margine: $\xi^t \geq 0$, richiedendo:

$$\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b) \geq 1 - \xi^t$$

Se $\xi^t = 0$, non ci sono problemi con $\mathbf{x}^t$. Se $0 < \xi^t < 1$, $\mathbf{x}^t$ è classificato correttamente, ma nel margine. Se $0 < \xi^t < 1$, $\mathbf{x}^t$ è classificato in modo errato.

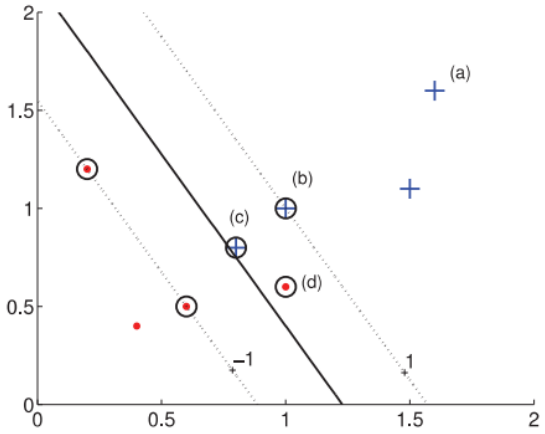


Figura 7: Possibili casi di classificazione delle istanze.

Si definisce quindi il *soft error*:

$$\sum_t \xi^t$$

E lo si aggiunge come termine di penalizzazione:

$$L_p = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_t \xi^t$$

C è il fattore di penalizzazione per avere un trade-off sulla complessità. Si penalizzano quindi sia i punti erroneamente classificati, sia i punti sul margine. Si ottiene così l'equazione:

$$L_p = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_t \xi^t - \sum_t \alpha^t [\mathbf{r}^t (\mathbf{w}^T \mathbf{x}^t + b) - 1 + \xi^t] - \sum_t \mu^t \xi^t$$

Dove μ^t sono i parametri lagrangiani per garantire la positività di ξ^t. Derivando rispetto ai parametri e ponendo il risultato uguale a 0 si ottiene:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_t \alpha^t r^t \mathbf{x}^t = 0 \qquad \Rightarrow \qquad \mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t$$

$$\frac{\partial L_p}{\partial b} = \sum_t \alpha^t r^t = 0$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0$$

Dato che $\mu^t \geq 0$, l'ultima uguaglianza implica che $0 \leq \alpha^t \leq C$. Si ottiene così l'espressione duale, che massimizziamo rispetto ad α^t:

$$L_p = \sum_t \alpha^t - \frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s$$

Rispettando i vincoli: $\sum_t \alpha^t r^t = 0$ e $0 \leq \alpha^t \leq C$, $\forall t$

Risolvendo ciò, le istanze che sono nella parte corretta dell'iperpiano con margine sufficiente hanno $\alpha^t = 0$. I support vector hanno $\alpha^t \geq 0$ e definiscono $\mathbf{w}$, di questi, quelli con $\alpha^t < C$ sono quelli che giacciono sul margine e vengono usati per calcolare b (hanno $\xi^t = 0$ e soddisfano $\mathbf{r}^t(\mathbf{w}^T\mathbf{x}^t + b) = 1$). Le istanze che sono nel margine o che sono erroneamente classificate hanno $\alpha^t = C$.

3.3 Kernel trick

Se il problema è non-lineare, esso può essere mappare in un nuovo spazio facendo una trasformazione non-lineare, in modo da usare un modello lineare nella nuova dimensione. Questa operazione può essere applicata sia nella regressione che nella classificazione; nel caso del Support Vector Machine questa operazione è nota come *kernel trick*.

Si suppone quindi di avere una nuova dimensione calcolata attraverso le funzioni

$$\mathbf{z} = \phi(\mathbf{x}) \qquad \text{dove } z_j = \phi_j(\mathbf{x}),\ j = 1,...,k$$

mappando dallo spazio di $\mathbf{x}$ d-dimensionale allo spazio di $\mathbf{z}$ k-dimensionale, dove scriviamo il discriminante come:

$$g(\mathbf{z}) = \mathbf{w}^T\mathbf{z}$$

$$g(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{j=1}^{k} w_j \phi_j(\mathbf{x})$$

Generalmente, k è maggiore di d e può essere anche maggiore di N, infatti è più vantaggiosa la forma duale dell'equazione, avendo quindi una complessità dipendente da N. Il problema è quindi lo stesso:

$$L_p = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_t \xi^t$$

a parte il fatto che i limiti sono definiti nel nuovo spazio:

$$r^t \mathbf{w}^T \phi(\mathbf{x}^t) \geq 1 - \xi^t$$

Si ottiene così l'equazione:

$$L_p = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_t \xi^t - \sum_t \alpha^t [r^t \mathbf{w}^T \phi^t(\mathbf{x}) - 1 + \xi^t] - \sum_t \mu^t \xi^t$$

Facendo poi le derivate rispetto ai parametri e imponendole uguali a 0, si ottiene:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} = \sum_t \alpha^t r^t \phi(\mathbf{x}^t)$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0$$

L'espressione duale diventa perciò:

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \phi(\mathbf{x}^t)^T \phi(\mathbf{x}^s)$$

Rispettando i vincoli: $\sum_t \alpha^t r^t = 0$ e $0 \leq \alpha^t \leq C, \ \forall t$

L'idea nel metodo *kernel machines* è quindi sostituire il prodotto scalare tra le funzioni di base, $\phi(\mathbf{x}^t)^T \phi(\mathbf{x}^s)$, con una funzione di kernel, $K(\mathbf{x}^t, \mathbf{x}^s)$. Perciò, invece di mappare due istanze $\mathbf{x}^t$ e $\mathbf{x}^s$ nello spazio di $\mathbf{z}$ e fare il prodotto scalare, si applica direttamente la funzione di kernel nello spazio di partenza.

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s K(\mathbf{x}^t, \mathbf{x}^s)$$

La funzione di kernel compare anche nel discriminante:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \sum_t \alpha^t r^t \phi(\mathbf{x}^t)^T \phi(\mathbf{x}) = \sum_t \alpha^t r^t K(\mathbf{x}^t, \mathbf{x})$$

Ciò implica che, se si ha una funzione di kernel, non c'è bisogno di mapparla in un nuovo spazio.

Quindi l'algoritmo del Support Vector Machine consiste nell'applicare l'ottimizzazione dell'espressione duale e applicare il kernel trick con una kernel function appropriata.

Di seguito un esempio per visualizzare l'operazione svolta dal kernel trick.

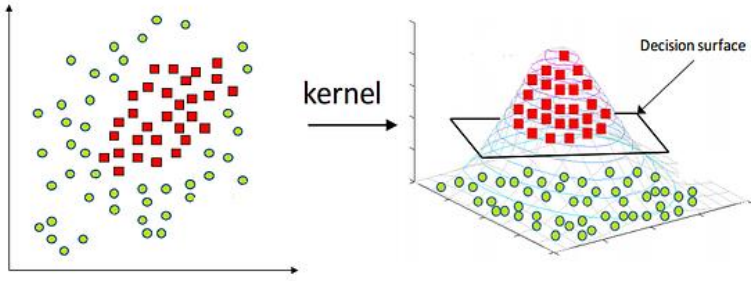


Figura 8: kernel trick applicato ad un caso con 2 attributi.

3.4 Kernel

Le funzioni di kernel più utilizzate sono: polinomiale, radiale di base e sigmoidale.

Una funzione polinomiale di grado q (q è un grado di libertà) è definita come:

$$K(\mathbf{x}^t, \mathbf{x}) = (\mathbf{x}^T \mathbf{x}^t + 1)^q$$

Prendendo $q = 1$ si ottiene un kernel lineare. Prendendo per esempio $q = 2$ si ottiene:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^2 = (x_1 y_1 + x_2 y_2 + 1)^2 = 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2$$

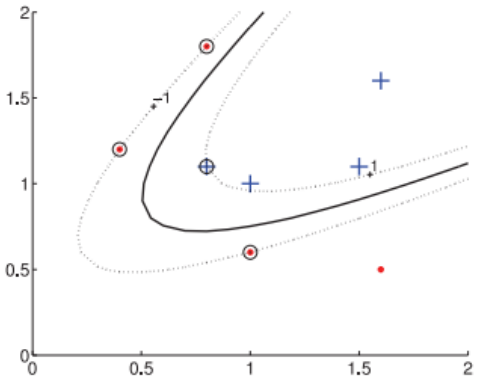


Figura 9: Discriminante e margine trovati da un kernel polinomiale di grado 2.

Una funzione radiale di base è definita come:

$$K(\mathbf{x}^t, \mathbf{x}) = \exp\left[-\frac{\| \mathbf{x}^t - \mathbf{x} \|^2}{2\sigma^2} \right]$$

ovvero un kernel sferico dove $\mathbf{x}^t$ è il centro e σ il raggio.

Infine, una funzione sigmoidale è definita come:

$$K(\mathbf{x}^t, \mathbf{x}) = \tanh\left(2\mathbf{x}^T \mathbf{x}^t + 1 \right)$$

Dove $\tanh(\cdot)$ ha la stessa forma di una funzione sigmoidea, eccetto che ha range tra -1 e +1.

3.5 Problema della definizione di un kernel

È possibile definire un kernel, e per fare ciò si parte dall'idea che esso sia una misura di similarità: $K(x, y)$ ha valori elevati se x e y sono simili.

Una funzione può essere usata come funzione di kernel solo se esiste ϕ tale che $K(x, y) = \phi(x)^T \phi(y)$

Per la definizione di un kernel è utile il *Teorema di Mercer*, in particolare un suo caso speciale. Considerando d punti $\left\{ x^1, ..., x^d \right\}$ e la matrice di kernel $K \in \mathbb{R}^{d \times d}$, con $K_{ij} = K(x^i, x^j)$, K è una funzione di kernel valida ($\exists \phi \mid K(x, y) = \phi(x)^T \phi(y)$) se e solo se per ogni punto $\left\{ x^1, ..., x^d \right\}$ la corrispondente matrice di kernel è definita positiva ($K \geq 0$).

Infatti, per ogni vettore z:

$$\begin{aligned}
\mathbf{z}^T K \mathbf{z} &= \sum_i \sum_j z_i K_{ij} z_j = \sum_i \sum_j z_i \left(\phi(x_i)^T \phi(x_j) \right) z_j = \sum_i \sum_j z_i \sum_k \left(\phi(x_i)_k \phi(x_j)_k \right) z_j = \\
&= \sum_i \sum_j z_i \sum_k \left(\phi(x_i)_k \phi(x_j)_k \right) z_j = \sum_i \sum_j z_i \sum_k \left(\phi(x_i)_k \phi(x_j)_k \right) z_j = \sum_i \sum_j z_i \sum_k \left(\phi(x_i)_k \phi(x_j)_k \right) z_j = \\
&= \sum \sum \sum z_i \left(\phi(x_i)_k \phi(x_j)_k \right) z_j = \sum_k \left(\sum_i z_i \phi(x_i)_k \right)^2 \geq 0
\end{aligned}$$

E quindi K è semidefinita positiva.

3.6 Kernel machines per la regressione

Si parte in primo luogo con un modello lineare:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

Come errore, per la regressione nel SVM si usa la funzione:

$$e_\varepsilon(r^t, f(x^t)) = \begin{cases} 0 & \text{se } |r^t - f(x^t)| < \varepsilon \\ |r^t - f(x^t)| - \varepsilon & \text{altrimenti} \end{cases}$$

che vuol dire tollerare errori fino ad ε, e che gli errori più grandi hanno un effetto lineare. Questa funzione di errore è più robusta al rumore (istanze molto separate). Si introduce quindi una variabile che tenga conto degli errori maggiori di ε:

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_t (\xi_+^t + \xi_-^t)$$

Tale che:

$$\begin{aligned} r^t - (\mathbf{w}^T\mathbf{x} + b) &\le \varepsilon + \xi_+^t \\ (\mathbf{w}^T\mathbf{x} + b) - r^t &\le \varepsilon + \xi_-^t \\ \xi_+^t, \xi_-^t &\ge 0 \end{aligned}$$

dove vengono usate due variabili per le deviazioni positive e negative. L'equazione con i parametri lagrangiani risulta:

$$\begin{aligned} L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_t (\xi_+^t + \xi_-^t) - \sum_t \alpha_+^t \left[\varepsilon + \xi_+^t - r^t + (\mathbf{w}^T\mathbf{x} + b)\right] \\ - \sum_t \alpha_-^t \left[\varepsilon + \xi_-^t + r^t - (\mathbf{w}^T\mathbf{x} + b)\right] - \sum_t (\mu_+^t \xi_+^t + \mu_-^t \xi_-^t) \end{aligned}$$

Facendo le derivate parziali e imponendole uguali a zero, si ottiene:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_t (\alpha_+^t - \alpha_-^t)\mathbf{x}^t = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_t (\alpha_+^t - \alpha_-^t)\mathbf{x}^t$$

$$\frac{\partial L_p}{\partial b} = \sum_t (\alpha_+^t - \alpha_-^t)\mathbf{x}^t = 0$$

$$\frac{\partial L_p}{\partial \xi_+^t} = C - \alpha_+^t - \mu_+^t = 0$$

$$\frac{\partial L_p}{\partial \xi_-^t} = C - \alpha_-^t - \mu_-^t = 0$$

Il duale è:

$$L_d = -\frac{1}{2}\sum_t \sum_s (\alpha_+^t - \alpha_-^t)(\alpha_+^s - \alpha_-^s)(\mathbf{x}^t)^T \mathbf{x}^s - \varepsilon \sum_t (\alpha_+^t - \alpha_-^t) + \sum_t r^t (\alpha_+^t - \alpha_-^t)$$

tale che:

$$0 \le \alpha_+^t \le C, \quad 0 \le \alpha_-^t \le C, \quad \sum_t (\alpha_+^t - \alpha_-^t) = 0$$

Una volta risolto ciò, si nota come tutte le istanze che sono nel tubo hanno $\alpha_+^t = \alpha_-^t = 0$.

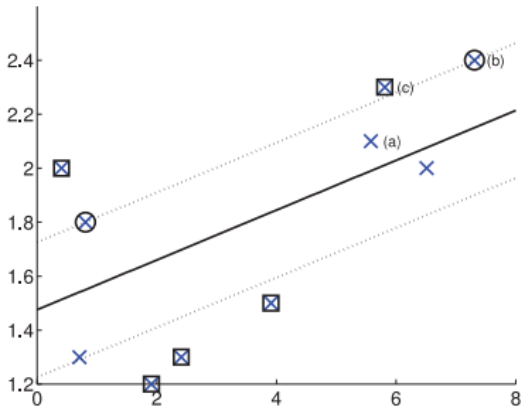


Figura 10: è mostrato un caso di regressione con discriminante lineare e il tubo di raggio $\varepsilon = 0.25$.

I support vector soddisfano $\alpha_+^t > 0$ o $\alpha_-^t > 0$ e sono di due tipi. Possono essere istanze sul confine del tubo (α_+^t o α_-^t compresi tra 0 e C), che vengono usate per calcolare b. Ci sono poi le istanze che sono fuori dal tubo e sono quelle con $\alpha_+^t = C$. Si può quindi scrivere la linea come somma pesata dei support vector:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_t (\alpha_+^t - \alpha_-^t)(\mathbf{x}^t)^T \mathbf{x} + b$$

Anche qui $(\mathbf{x}^t)^T \mathbf{x}$ può essere sostituito con un kernel $K(\mathbf{x}^t, \mathbf{x})$ per avere un modello non lineare.

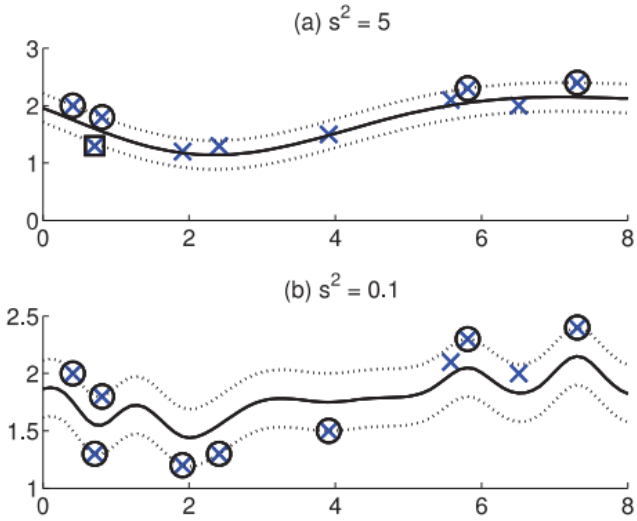


Figura 11: Discriminante con kernel Gaussiano, prendendo due diversi valori per $\varepsilon = 0.25$.

3.7 Caso applicativo e confronto con regressione logistica

Prendendo in analisi lo stesso dataset utilizzato per la regressione logistica, si vuole ora eseguire un algoritmo di *support vector machine*. Per quanto riguarda il codice Python, è necessario riscrivere solamente le ultime righe: quelle in cui viene definito il classificatore in grado di predire la classe delle istanze presenti nel set di verifica. Di seguito il cambiamento:

```
#Modeling (SVM with Scikit-Learn)
from sklearn import svm

# SVC = support vector classifiers
classifier = svm.SVC(kernel='linear', gamma='auto', C=2)
classifier.fit(X_train, y_train)

y_predict = classifier.predict(X_test)

#Evaluation (results)

print(classification_report(y_test, y_predict))
```

La tipologia di Kernel utilizzata è quella lineare. Infatti, avendo risultati molto buoni con l'algoritmo di regressione logistica, si hanno risultati altrettanto validi utilizzando un Kernel lineare con SVM. Questo perché le due classi del dataset sono linearmente separabili nella dimensione di input, a parte qualche caso. I risultati ottenuti possono essere visualizzati nel seguente report:

	Precision	Recall	F1-score	Support
Benigno	1.00	0.94	0.97	90
Maligno	0.90	1.00	0.95	47
Accuracy			0.96	137

Tabella 5: Risultati ottenuti con un kernel lineare

Si può migliorare leggermente la prestazione del classificatore utilizzando un Kernel polinomiale di quinto grado. Si ottengono migliori risultati in quanto si riesce a migliorare la qualità del classificatore di riconoscere i casi più incerti, ma a costo di un tempo computazionale maggiore. Per prevenire l'*overfitting*, viene aumentata la penalità sulla complessità del modello (C=5).

```
#Modeling (SVM with Scikit-Learn)
from sklearn import svm

# SVC = support vector classifiers
classifier = svm.SVC(kernel='poly', degree=5, gamma='auto', C=5)

classifier.fit(X_train, y_train)

y_predict = classifier.predict(X_test)

#Evaluation (results)

print(classification_report(y_test, y_predict))
```

	Precision	Recall	F1-score	Support
Benigno	1.00	0.97	0.98	90
Maligno	0.94	1.00	0.97	47
Accuracy			0.98	137

Tabella 6: Risultati ottenuti con un kernel polinomiale di quinto grado

Si ha quindi che gli algoritmi di *support vector machine* e regressione logistica sono entrambi molto performanti sul dataset preso in esame. Le prestazioni migliori si ottengono con il *support vector machine* utilizzando un kernel polinomiale, ma ciò risulta necessario solo a fronte di una richiesta di precisione molto elevata. Perciò, conviene utilizzare la regressione logistica se non è richiesta una tale precisione in quanto essa risulta computazionalmente vantaggiosa, caratteristica di molto importante nel caso il numero di istanze di ingresso sia particolarmente elevato.

CONCLUSIONE

All'interno di questo documento, dopo una breve narrazione sulla storia e le applicazioni del machine learning, sono stati trattati i metodi di approccio alla scelta di un modello in maniera generale, definendo i parametri su cui possono essere basate le decisioni di progetto e le procedure che si ripetono nei vari algoritmi.

Successivamente, sono stati trattati due algoritmi di apprendimento supervisionato: la regressione logistica e il *support vector machine*. Di tali modelli sono state presentate le caratteristiche matematiche e le interpretazioni per utilizzarli al fine di effettuare diagnosi di un tumore. È stato preso in esame un dataset di pazienti con determinate caratteristiche, conoscendo la diagnosi effettiva, per essere in grado di predire nuovi casi; a tale scopo è stato utilizzato un set di verifica pari al 20% del numero di pazienti. L'algoritmo di machine learning è stato sviluppato utilizzando prima la regressione logistica e successivamente il *support vector machine*. Si è notato come entrambi i modelli riescano ad ottenere ottimi risultati, con *accuracy* maggiore del 96%, infatti le due classi del problema in analisi sono linearmente separabili se non per qualche eccezione.

Dall'elaborazione dell'algoritmo di regressione logistica è emerso che esso sia un metodo di semplice implementazione ed interpretazione; il modello è inoltre efficiente dal punto di vista computazionale e non richiede grandi quantità di risorse. Tuttavia, la regressione logistica presenta alcuni limiti, ad esempio può avere difficoltà nel gestire dati non linearmente distribuiti e può soffrire di problemi di *overfitting*.

L'analisi dei dati attraverso l'algoritmo di *support vector machine* ha fatto emergere come esso sia in grado di gestire anche dati non distribuiti linearmente grazie all'utilizzo di Kernel e che presenti una maggiore robustezza all'*overfitting* intrinseca al modello. Inoltre, questo modello è in grado di rappresentare funzioni non lineari complesse e in dimensioni molto elevate, avendo di conseguenza una scarsa interpretabilità dei risultati. Un altro limite dell'algoritmo di *support vector machine* è la complessità computazionale, infatti, i tempi di elaborazione dei dati crescono con l'aumento della complessità del modello.

In conclusione, non si può definire un algoritmo migliore fra i due presi in analisi: un modello viene preferito ad un altro in base all'ambito di applicazione, al caso preso in analisi e alle specifiche richieste, come complessità, efficienza e precisione.

BIBLIOGRAFIA

- Asuncion, Newman, UCI Machine Learning Repository, 2007.
- A. Ng, “Stanford CS229: Machine Learning”, Autumn 2018 Lectures.
- E. Alpaydin, “Introduction to Machine Learning”, The MIT Press, quarta edizione.
- S. Russell, P. Norvig, “Intelligenza artificiale. Un approccio moderno”, Pearson, terza edizione.
- P. Wentworth, J. Elkner, A. B. Downey, C. Meyers, “How to Think Like a Computer Scientist: Learning with Python 3”, febbraio 2019.
- D. Cournapeau, M Brucher, “Logistic regression”, 2007-2023, Scikit-learn, www.scikit-learn.org
- D. Cournapeau, M Brucher, “Support Vector Machine”, 2007-2023, Scikit-learn, www.scikit-learn.org
- C. Cortes, V. Vapnik, “Support-Vector Networks”, Kluwer Academic Publishers, Boston, 1995.