UNIVERSITÀ DI PISA

Dipartimento di Informatica

MACHINE LEARNING

Appunti dalle lezioni del Prof. Alessio Micheli

Massimo Bilancioni

Guido Narduzzi

Filippo Quattrocchi

Francesco Zigliotto

ANNO ACCADEMICO 2019–2020

INDICE

1 INTRODUZIONE

1.1 LEZIONE DI GIOVEDÌ 26 SETTEMBRE

1.1.1 Contestualizzazione

L'obiettivo del Machine Learning è insegnare ad un sistema un compito preciso, costruendo un modello utilizzabile per predire il corretto output, dopo aver esaminato un gran numero di esempi. Tale metodo di apprendimento è detto *per generalizzazione*.

È utile per esempio quando l'approccio teorico a un determinato problema è difficilmente praticabile, o quando i dati in input sono poco accurati, affetti da errore o incompleti.

Osservazione 1.1. Il Machine Learning non consente di prevedere fenomeni *casuali* (come i numeri della lotteria). Inoltre non permette di inferire relazioni di *causalità* tra due fenomeni tramite la sola analisi dei dati.

Definizione 1.2. (Machine Learning). Il *Machine Learning* studia e propone metodi per inferire funzioni o correlazioni che, a partire da dati osservati, producano un buon *fit* dei *samples* forniti, generalizzandoli con ragionevole accuratezza.

1.1.2 Terminologia

Un *machine learning system* si compone di *dati*, *tasks*, *modelli*, *algoritmi di apprendimento* e *validazione*.

Definizione 1.3. (Dati). I *dati* rappresentano le *esperienze disponibili*. Possono essere organizzati in un certo numero l di istanze $\boldsymbol{x_p}$ (*samples*, *instances*), ciascuno contenente n attributi (*features*). Con $x_{p,j}$ indicheremo l'attributo j-esimo della p-esima istanza.

Osservazione 1.4. Se un attributo può assumere un numero finito di valori, risulta spesso conveniente rappresentarlo come un vettore di dimensione k (dove k è il numero dei valori possibili) con componenti tutte nulle a parte una.

Esempio 1.5. Se i valori possibili sono i tre colori *rosso* (R), *verde* (G), *blu* (B), si pone

$$R = (1, 0, 0), \quad G = (0, 1, 0), \quad B = (0, 0, 1).$$

Definizione 1.6. (Rumore, outliers). Il *rumore* è l'aggiunta di fattori esterni dovuta al processo di misura (e non alla legge soggiacente). Gli *outliers* sono dati che si collocano molto lontani rispetto agli altri.

I *tasks* sono in genere di due tipologie (ma ce ne sono altre):

Definizione 1.7. (Supervised learning). Nel *supervised learning* sono dati dei *samples* di una funzione f ignota, nella forma

$$\langle input,\ output \rangle$$

si tratta di trovare una buona approssimazione di f. Gli input sono detti anche *variabili indipendenti*, gli ouput *variabili dipendenti* o *risposte*. Se f è a valori discreti, il problema si dice di *classificazione*. Se l'output è costituito da valori reali, allora si parla di *regressione*.

Definizione 1.8. (Unsupervised learning). Nell'*unsupervised learning* non si dispone di input e output nelle istanze, ma solo di dati non etichettati. Un problema tipico è quello di raggruppare tali dati secondo determinati criteri.

Noi ci occuperemo soprattutto di supervised learning.

Definizione 1.9. (Modello, ipotesi). Il modello cerca di descrivere la relazione tra i dati con un *linguaggio*, legato alla rappresentazione dei dati. Le *ipotesi* sono le funzioni h_w proposte dal modello per approssimare la "vera" funzione f. Le ipotesi sono indicizzate da parametri (w) e formano uno *spazio delle ipotesi* H.

In generale non esiste un modello *ottimo*: se un modello di apprendimento è il migliore in qualche problema, sarà peggiore di altri in altri problemi. Questo concetto è noto come *No Free Lunch Theorem*, ovvero "non c'è un pranzo gratis" (mah, sarà qualche detto inglese, ndr). In ogni caso, questo non significa che tutti i modelli siano equivalenti.

Definizione 1.10. (Algoritmo di apprendimento). Un *algoritmo* di apprendimento si occupa di cercare nello spazio delle ipotesi H (di un modello fissato) la migliore approssimazione della funzione f.

Come definiamo *buona approssimazione*? Si utilizza una *loss function* $L\big(h(x), d\big)$, che misura la distanza tra $h(x)$ e d, dove d è il valore osservato (cioè $f(x)$ più eventualmente il rumore).

Definizione 1.11. (Errore). L'errore è definito da

$$E = \frac{1}{l} \sum_{i=1}^{l} L\big(h(x_i), d_i\big) \tag{1.1}$$

dove x_i sono le istanze e $d_i = f(x_i)$ i valori osservati.

Esempio 1.12. Nei problemi di regressione spesso come loss function si usa

$$L\big(h(x), d\big) = \big(d - h(x)\big)^2 \tag{1.2}$$

e in tal caso l'errore si dice *errore quadratico medio* (MSE).

Se siamo di fronte a un problema di classificazione, allora è più conveniente usare la loss function che vale 1 se i suoi due argomenti sono uguali (e dunque la classificazione è corretta) e 0 altrimenti.

Osservazione 1.13. In Machine Learning, quando si parla di *performance*, si fa riferimento all'accuratezza predittiva, non all'efficienza computazionale.

1.2 LEZIONE DI VENERDÌ 27 SETTEMBRE

1.2.1 Fitting e overfitting

Definizione 1.14. (Validazione). La *validazione* valuta la capacità di generalizzazione di una determinata ipotesi, misurandone l'accuratezza.

In generale non possiamo assumere che se un'ipotesi h approssima bene la funzione f sui samples di allenamento, allora h approssima f anche su nuove istanze. C'è per esempio il problema dell'*overfitting*.

Definizione 1.15. (Overfitting). L'*overfitting* avviene quando sottostimiamo l'errore sperimentale nel fitting e quindi aumenta il *vero* errore sui dati sconosciuti. L'overfitting avviene quindi se il modello è troppo complesso ed è perciò in grado di *fittare il rumore*.

Esempio 1.16. (Fitting polinomiale). Supponiamo di avere una funzione reale e di voler risolvere il problema di regressione con un *fit polinomiale*. Le ipotesi sono della forma

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=0}^{M} w_i x^i \tag{1.3}$$

dove $\boldsymbol{w}$ è il vettore dei coefficienti w_i. Si cerca di minimizzare l'errore quadratico medio. Si osserva che se il polinomio ha grado troppo basso i *training samples* non vengono fittati correttamente (*underfitting*). Aumentando troppo il grado del polinomio nel modello, l'errore sui *training samples* diminuisce, mentre quello sui *test samples* aumenta (e i coefficienti del polinomio aumentano di molto in modulo): questo comportamento è tipico dell'*overfitting*.

Infine, a parità di grado, si nota che aumentando il numero di dati il fitting migliora molto, indipendentemente dal fatto che ci sia molto rumore.

1.2.2 Setting semplificato

Formalmente, quindi, disponiamo di

- una funzione f ignota da approssimare;
- un modello con un relativo spazio di ipotesi H;
- una *loss function* L;
- una distribuzione di probabilità $P(x, d)$ per lo spazio dei dati, dove $d(x)$ corrisponde alla distribuzione dei dati sperimentali ad x fissato (d corrisponde al valore di $f(x)$ misurato sperimentalmente, quindi affetto da rumore, e per lo stesso x naturalmente si possono avere più misure).

L'obiettivo teorico sarebbe minimizzare il *rischio*, cioè il *vero errore su tutti i dati*:

$$R = \int L\big(h_w(x), d\big)\, dP(x, d). \tag{1.4}$$

È tuttavia più praticabile lavorare con il *rischio empirico*, cioè l'errore sui *training samples*:

$$R_{\text{emp}} = \frac{1}{l} \sum_{i=1}^{l} L\big(h_w(x_i), d_i\big). \tag{1.5}$$

Chiaramente non bisogna procedere scegliendo il modello che minimizzi il rischio empirico, per via dell'overfitting. Occorre invece scegliere il modello considerando insieme il rischio empirico e la complessità.

Si può mostrare che, con probabilità $1 - \delta$, vale

$$R \leq R_{\text{emp}} + \varepsilon(1/l, VC, 1/\delta) \tag{1.6}$$

dove l è il numero di samples, VC è la "complessità del modello" (il *grado* del polinomio, nell'esempio del fit polinomiale, in generale è la dimensione di Vapnik-Chervonenkis) ed ε è un'opportuna funzione: solitamente si assume che ε sia direttamente proporzionale ai suoi due primi argomenti. Infatti, per grandi valori di l, il rischio R è minore. Mentre per grande complessità del modello il rischio empirico R_{emp} è minore ma il rischio R può aumentare, per via dell'overfitting.

1.2.3 Validazione

La validazione è composta da due fasi:

- la selezione del modello (*model selection*) si occupa di scegliere il modello più adatto a trattare il problema;
- il giudizio del modello (*model assessment*) valuta la capacità predittiva del modello su *test samples*.

In particolare si divide l'insieme di dati in TR (*training set*), VL (*validation set*) e TS (*test set*). A questo punto avviene la *model selection*, in cui

- si fissa un determinato modello e si usano i dati in TR per trovare la funzione h che minimizzi il rischio empirico;
- si usano i dati in VL per determinare la bontà del modello;
- si cicla sui due step sopra al fine di determinare il modello migliore.

Una volta ottenuto il modello migliore (già pronto per l'utilizzo, con tutti parametri fissati dai dati in TR), si effettua il *model assessment* e si valuta il comportamento sui nuovi dati presenti nel TS.

Esempio 1.17. (k-fold cross-validation). Per esempio, si può dividere l'insieme di dati per la *model selection* in k sottoinsiemi $D_1, \ldots, D_k$ e poi, per ogni $j = 1, \ldots, k$ utilizzare D_j come VL e l'unione di tutti gli altri D_i come TR. La bontà del modello sarà quindi data dalla "media delle bontà" calcolate al variare di j.

2 CONCEPT LEARNING

2.1 LEZIONE DI MARTEDÌ 1 OTTOBRE

2.1.1 Concept learning

Definizione 2.1. (Apprendimento). Secondo il Mitchell: migliorare nel task T, rispetto alla performance di misura P, basandosi sull'esperienza E.

Di base il *concept learning* è il problema di trovare una funzione che restituisca la classe (assunta) di x. Quindi il concept learning consiste nell'approssimare una funzione booleana.

Diciamo che $h : X \to \{0, 1\}$ *soddisfa* x se $h(x) = 1$. Inoltre un'ipotesi h è *consistente* con un esempio $\big(x, c(x)\big)$ se $h(x) = c(x)$. Diciamo che h è *consistente* con un set D se

$$\big(x, c(x)\big) \in D \implies h(x) = c(x). \tag{2.1}$$

Il numero di possibili funzioni booleane con n input binari è 2^{2^n} quindi non ci possiamo lavorare davvero. Dobbiamo necessariamente lavorare con uno spazio delle ipotesi ristretto, anche se questo introduce un bias considerevole. In particolare vedremo come farlo con semplici regole congiuntive e funzioni lineari.

Lo spazio H di tutte le funzioni h generate da *congiunzione* di positive literals[1] ha cardinalità 2^n. Ad esempio, delle possibili funzioni h sono $h(x) = (x_1 \wedge x_5)$ o $h(x) = x_3$, ma anche $h(x) = \text{true}$. Invece lo spazio delle funzioni generate da congiunzione di literals (anche negati) sono $3^n + 1$. Un esempio è $h(x) = (x_1 \wedge \neg x_7)$.

In questo caso ogni ipotesi h è la congiunzione di un certo numero di vincoli sugli attributi, ognuno dei quali può essere

- l'assegnazione di un valore specifico (per esempio: "acceso=true"),
- un'accettazione di qualsiasi valore (si indica con ?, per esempio: "acceso=?"),
- il valore vuoto o *ipotesi nulla* (si indica con $\emptyset$ e vuol dire $x_i \wedge \neg x_i$, per esempio: "acceso non ammette alcun valore ammissibile").

Il modo in cui rappresentiamo lo spazio delle ipotesi determina come avviene la ricerca della h che meglio approssima c. Dobbiamo pertanto tentare di strutturalo in modo comodo per cercare rapidamente. Per esempio, date due ipotesi, una potrebbe essere strettamente più generale dell'altra (i constraints sono uguali se presenti in entrambe e la seconda ne ha almeno uno in meno).

Definizione 2.2. (Ordinamento per generalità). Diremo che h_j è *più generale* di h_k se e solo se, per ogni $x \in X$, vale

$$h_k(x) = 1 \implies h_j(x) = 1. \tag{2.2}$$

[1] Un positive literal è una proposizione atomica x. Un literal è un positive literal x o la sua negazione $\neg x$.

Chiaramente due ipotesi non sempre sono comparabili, quindi l'ordinamento appena introdotto è parziale. Possiamo usarlo per organizzare efficientemente la ricerca in H. Può essere utile cercare di visualizzare lo spazio delle ipotesi. In questo caso possiamo costruire un grafo diretto dove un nodo (ipotesi) è figlio di un altro se è più specifico. Possiamo partire da un'ipotesi specifica e usare l'ordine parziale per muoverci sul grafo di un lato alla volta, fino a trovarci in ipotesi consistenti con i dati e generalizzando il meno possibile.

2.1.2 Find-S algorithm

L'algoritmo *Find-S* funziona come segue.

1. Si inizializza h all'ipotesi che assegna $\emptyset$ a ogni attributo. Così facendo, h è massimamente specifica (minimamente generale).
2. Per ogni istanza positiva x e per ogni attributo a_i in h:
   - se a_i è soddisfatto da x, non si fa nulla;
   - altrimenti si sostituisce a_i in h col constraint appena meno specifico che è soddisfatto da x.
3. Si restituisce h.

Proprietà: in uno spazio di ipotesi descritto da congiunzioni di attributi trova l'ipotesi più specifica consistente con gli esempi di training positivi e sarà anche consistente con i negativi, posto che c sia in H (visto che $c \geq h$). Però la nostra imposizione su H è pesante, inoltre l'algoritmo di suo non è un granché. Infatti, visto che ignora i valori negativi non capisce se i dati non sono consistenti (quindi non è robusto al rumore). Inoltre non c'è davvero un buon motivo per volere l'ipotesi consistente più specifica, e peraltro potrebbe esserci più di una ipotesi più specifica consistente (non comparabili tra loro).

Si introducono quindi i *version spaces* con l'idea di ottenere una descrizione dell'insieme di tutte le h consistenti con D.

Definizione 2.3. (Version space). Un *version space* relativo allo spazio delle ipotesi H e al training set D è il sottoinsieme delle ipotesi consistenti con tutti i training examples. Si indica con $VS_{H,D}$.

Un primo modo per ottenere una descrizione di $VS_{H,D}$ è quella del *List-Then Eliminate Algorithm* che opera semplicemente inizializzando ad H il version space e poi analizzando i singoli training example per rimuovere, volta per volta, le ipotesi ad essi incoerenti. Quello che si ottiene alla fine è quindi la lista completa degli elementi di $VS_{H,D}$.

Altrimenti si può pensare di rappresentare $VS_{H,D}$ in maniera più compatta per mezzo dei concetti di *general boundary* e *specific boundary*.

Definizione 2.4. (General boundary). Il *general boundary* G di $VS_{H,D}$ è l'insieme dei membri di H consistenti con D massimamente generali.

Definizione 2.5. (Specific boundary). Il *specific boundary* S di $VS_{H,D}$ è l'insieme dei membri di H consistenti con D massimamente specifici.

Vale infatti il seguente teorema.

Teorema 2.6. *Ogni membro di $VS_{H,D}$ si trova tra questi insieme di limitazione. Più propriamente vale*

$$VS_{H,D} = \{h \in H : \exists s \in S \; \exists g \in G \; s \leq h \leq g\}. \tag{2.3}$$

L'algoritmo che permette di ottenere G e S è il *candidate elimination algorithm*, che è simile al Find-S. Viene omessa la sua descrizione perché non di particolare interesse per questo corso.

2.1.3 Il bias induttivo e il suo ruolo

Il bias sta nel fatto che assumiamo che c stia nel nostro spazio H, ovvero che sia rappresentabile con le regole che abbiamo deciso a priori (in questo caso che sia rappresentabile come una congiunzione di literals). Ad esempio il nostro spazio è incapace di rappresentare una semplice disgiunzione. Se volessimo un *unbiased learner* dovremmo scegliere come H l'insieme potenza $\mathcal{P}(X)$, e comunque classificheremmo in modo ambiguo ogni elemento fuori dal training set. Quindi l'unbiased learner non è in grado di generalizzare (non ce ne facciamo niente). Il bias è dunque fondamentale per poter apprendere, però non sappiamo a priori quale bias sia il migliore per la generalizzazione. Dobbiamo caratterizzarlo per diversi approcci di apprendimento, per capire quando un certo bias sarà appropriato.

Definizione 2.7. (Bias induttivo). Dati

- un algoritmo di concept learning L,
- un insieme di istanze X,
- una funzione target c,
- un insieme di training examples $D_c := \{(x, c(x)) : x \in \tilde{X}\}$ con $\tilde{X} \subseteq X$,

indichiamo con $L(x, D_c)$ la classificazione assegnata a x da L dopo il training su D_c.

Definiamo un *bias induttivo* (*inductive bias*) di L come un qualsiasi insieme minimale (rispetto all'inclusione) B di asserzioni tali che, per ogni target concept c e insieme dei training examples D_c valga

$$B \wedge D_c \wedge x \vdash L(x, D_c) \tag{2.4}$$

per ogni $x \in X$.[2]

Con questa definizione è possibile associare a un *sistema induttivo*, con bias induttivo B, un *sistema deduttivo* equivalente. Per esempio, dato H spazio delle ipotesi, definiamo l'algoritmo L come quello che con un input (x, D_c) restituisce:

- se tutte le ipotesi in VS_{H,D_c} danno risposta unanime all'input x, tale risposta,
- nessun output altrimenti.

Questo sistema è induttivo con bias $B = (c \in H)$. Un sistema deduttivo (*theorem prover*) restituisce lo stesso output se gli forniamo esplicitamente il presupposto B.

[2] Con $\phi \vdash \psi$ si intende che ψ segue logicamente da ϕ.

2.1.4 Esempi di bias

L'algoritmo appena descritto è, di fatto, il candidate elimination algorithm (se H è lo spazio generato dalla congiunzione di attributi). Abbiamo appena visto che il suo bias è $B = (c \in H)$.

L'*apprenditore di Rote* funziona classificando x se e solo se lo ha già visto (cioè x è un training sample). Esso non introduce bias, e infatti non generalizza.

L'algoritmo Find-S ha il seguente bias: per ogni training set D, la funzione c si trova sullo specific boundary di $VS_{H,D}$.

Per superare le restrizioni date dall'uso del solo *and* si possono usare modelli più flessibili. Ad esempio, nel caso di H discreto: decision trees (Mitchell cap.3), genetic algorithms, inductive logic programming (Mitchell cap.10).

I *decision trees* rappresentano una disgiunzione delle congiunzioni dei constrains sul valore degli attributi. Sono utili perché permettono di lavorare su un H *completo* (cioè H contiene tutte le 2^{2^n} funzioni) facendo una *ricerca incompleta*. Questo dà un modello flessibile che non rischia di escludere a priori il target, ma che per la sua flessibilità si porta dietro il rischio di overfitting.

Bibliografia: Mitchell capp. 1 & 2, AIMA 19.1.

3 LINEAR MODELS

3.1 LEZIONE DI GIOVEDÌ 3 OTTOBRE

3.1.1 Modelli lineari

I modelli lineari sono utili per risolvere problemi di supervised learning, ossia problemi in cui dato un certo numero di coppie $(x, f(x))$, dove la funzione f è ignota, si cerca una buona approssimazione di f.

Se f è una funzione discreta, siamo in presenza di un problema di classificazione, se invece è a valori in $\mathbb{R}^n$, abbiamo un problema di regressione.

Regressione lineare

Dato un insieme di l coppie di reali (x_p, y_p), cerchiamo una funzione del tipo $h_{\boldsymbol{w}}(x) = w_1 x + w_0$ che interpoli al meglio i dati iniziali. In altre parole vogliamo trovare il vettore $\boldsymbol{w} = (w_1, w_0)$ che minimizzi una certa funzione $E(\boldsymbol{w})$ che chiameremo errore o Loss.

Se utilizziamo il metodo LMS (*least mean square*), come funzione E scegliamo:

$$E(\boldsymbol{w}) = \sum_{p=1}^{l} \left(y_p - h_{\boldsymbol{w}}(x_p)\right)^2 \tag{3.1}$$

Per risolvere questo problema possiamo cercare i punti stazionari come zeri di ∇E, infatti poiché E è una funzione convessa (è somma di funzioni convesse), i suoi punti stazionari saranno dei minimi.

Questo problema si può affrontare in modo analogo anche se i $\boldsymbol{x_p}$ sono vettori di $\mathbb{R}^n$, in questo caso la funzione interpolante sarà della forma $h(\boldsymbol{x}) = \boldsymbol{w_1}^T \boldsymbol{x} + w_0$.

Esercizio: Nel caso in cui $x_p \in \mathbb{R}$ per ogni p, mostrare che $w_1 = \frac{\mathrm{Cov}[x,y]}{\mathrm{Var}[x]}$ e $w_0 = \overline{y} - w_1 \overline{x}$ ($\overline{x}$ indica la media di x).

Problemi di classificazione

Abbiamo 2 insiemi di punti in $\mathbb{R}^n$ generati con due distribuzioni ignote, vogliamo trovare un modo per prevedere a quale dei due insiemi apparterranno dei punti generati in futuro. Questo può essere visto come un problema di supervised learning in cui f vale 0 se il punto appartiene al primo insieme e 1 se appartiene al secondo.

Questo problema può essere affrontato con un modello lineare dividendo lo spazio con un iperpiano, dunque definendo $h(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w_1}^T \boldsymbol{x} + w_0)$. In molti casi l'iperpiano in grado di separare i due insiemi di punti non è unico, ma le possibilità possono essere ridotte aumentando le dimensioni del training set.

Approccio diretto

Torniamo al caso in cui $h(\boldsymbol{x})$ è una funzione lineare, dunque $h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$. Per usare il metodo dei minimi quadrati, minimizziamo la funzione $E(\boldsymbol{w}) = \sum_{p=1}^{l}(y_p - \boldsymbol{x_p}^T\boldsymbol{w})^2 = \|\boldsymbol{y} - X\boldsymbol{w}\|^2$. Poiché E è una funzione quadratica, sappiamo che esistono dei minimi locali, per trovarli cerchiamo gli zeri di ∇E. Vale

$$\frac{\partial E}{\partial w_j} = \sum_{p=1}^{l}\frac{\partial}{\partial w_j}(y_p - \boldsymbol{x_p}^T\boldsymbol{w})^2 = -2\sum_{p=1}^{l}(y_p - \boldsymbol{x_p}^T\boldsymbol{w})(\boldsymbol{x_p})_j, \tag{3.2}$$

quindi abbiamo bisogno che

$$\sum_{p=1}^{l}(y_p - \boldsymbol{x_p}^T\boldsymbol{w})(\boldsymbol{x_p})_j = 0, \tag{3.3}$$

ovvero

$$X^T\boldsymbol{y} = X^TX\boldsymbol{w}. \tag{3.4}$$

Dove X è la matrice che ha per righe i vettori $\boldsymbol{x_p}^T$ e $\boldsymbol{y}$ è il vettore degli y_p. Se X^TX è invertibile, la soluzione è unica ed è $\boldsymbol{w} = (X^TX)^{-1}X^T\boldsymbol{y}$, altrimenti esistono infinite soluzioni (visto che il minimo esiste sicuramente). La matrice $(X^TX)^{-1}X^T$ è detta *pseudoinversa di Moore-Penrose* di X e si indica con X^+.

3.2 LEZIONE DI VENERDÌ 4 OTTOBRE

3.2.1 Metodo del gradiente

Definizione 3.1. (Metodo del gradiente steepest descent). Il *metodo del gradiente steepest descent* è un algoritmo iterativo per trovare punti di minimo (locale) della funzione E al variare dei parametri $\boldsymbol{w}$. Esso consiste nel fissare un punto $\boldsymbol{w}_{\mathbf{old}}$ nel dominio di E e muoversi nella direzione opposta a quella del gradiente, ottenendo

$$\boldsymbol{w}_{\mathbf{new}} = \boldsymbol{w}_{\mathbf{old}} - \eta\nabla E(\boldsymbol{w}_{\mathbf{old}}) = \boldsymbol{w}_{\mathbf{old}} + \eta\Delta\boldsymbol{w} \tag{3.5}$$

dove $\eta > 0$ è lo *step-size parameter* e $\Delta\boldsymbol{w} := -\nabla E(\boldsymbol{w}_{\mathbf{old}})$.

Nel calcolo del gradiente di E si può procedere in vari modi.

- *Metodo batch*: siccome disponiamo di una famiglia di l esempi, possiamo considerare E come la media degli errori ottenuti sui singoli samples (come al solito), e quindi
$$E(\boldsymbol{w}) = \frac{1}{l}\sum_{p=1}^{l}(y_p - \boldsymbol{w}^T\boldsymbol{x_p})^2. \tag{3.6}$$
- *Metodo on-line*: alternativamente, è possibile procedere separatamente sui vari samples, cioè applicare l successive iterazioni del metodo del gradiente, dove l'i-esima iterazione è applicata alla funzione
$$E_p(\boldsymbol{w}) = (y_p - \boldsymbol{w}^T\boldsymbol{x_p})^2. \tag{3.7}$$

Questo secondo metodo rende computazionalmente più veloce l'applicazione di ogni iterazione, ma al tempo stesso l'avvicinamento al punto di minimo avviene in modo meno diretto.

Naturalmente bisogna scegliere opportunamente il parametro η in modo da garantire un buon compromesso tra stabilità e velocità di convergenza dell'algoritmo. Per una maggiore efficienza si può scegliere il parametro η dinamicamente ad ogni iterazione, in modo da ottenere spostamenti grandi se l'errore è grande, e viceversa.

3.2.2 Espansione in base lineare

In molti casi, però, i modelli lineari non sono sufficienti a descrivere accuratamente il problema. Ci occuperemo ora di modelli non lineari rispetto alla variabile $\boldsymbol{x}$, ma lineari rispetto a $\boldsymbol{w}$.

Definizione 3.2. (Espansione in base lineare). Un'espansione in base lineare è una funzione

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{k=1}^{K} w_k \phi_k(\boldsymbol{x}) \tag{3.8}$$

dove ϕ è una qualunque funzione $\phi : \mathbb{R}^n \longrightarrow \mathbb{R}^K$ e $K \geq n$.

Esempio 3.3. I polinomi sono un esempio di espansione in base lineare (in dimensione 1), in cui $\phi_k(x) = x^k$.

Nella scelta della funzioni ϕ bisogna considerare che dietro la grande espressività consentita da funzioni poco lineari si annida il rischio dell'overfitting.

3.2.3 Regolarizzazione di Tikhonov

Per evitare l'overfitting risulta conveniente aggiungere un termine di *penalizzazione* al costo E in modo da evitare che i pesi possano assumere valori (in modulo) troppo grandi. Si pone dunque

$$E(\boldsymbol{w}) = \frac{1}{l} \sum_{p=1}^{l} (y_p - \boldsymbol{w}^T \boldsymbol{x_p})^2 + \lambda \|w\|_2^2 \tag{3.9}$$

dove il parametro $\lambda > 0$ consente di scegliere quanto peso dare a tale penalizzazione.

In termini di equazioni normali, si ottiene

$$\boldsymbol{w} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}, \tag{3.10}$$

mentre in termini di metodo del gradiente si ha

$$\boldsymbol{w_{\text{new}}} = \boldsymbol{w_{\text{old}}} + \eta \Delta \boldsymbol{w} - 2\lambda \boldsymbol{w_{\text{old}}}. \tag{3.11}$$

Osservazione 3.4. È interessante notare che in questo modo il modello dipende solo da un parametro, cioè il parametro di regolarizzazione λ. La scelta del λ migliore è compito del *model selection*.

4 ALGORITMO K-NN

4.1 LEZIONE DI MARTEDÌ 8 OTTOBRE

4.1.1 k-Nearest Neighbours

Definizione 4.1. Il *k-Nearest Neighbours* è un algoritmo per la classificazione di oggetti che si basa sul confronto con oggetti *vicini* (con un'opportuna definizione di distanza) a quello considerato per i quali è stata già fornita la corretta classificazione.

Nel k-NN non si costruisce un'ipotesi esplicita a partire dai training data (TD), ma si prende di volta in volta il nuovo punto da classificare e, confrontandolo con i TD, gli si assegna un valore.

Esempio 4.2. (1-NN). Per $k = 1$, l'algoritmo è il seguente:

- si memorizzano gli l dati di training $(\boldsymbol{x_p}, y_p)$, con $p = 1, \dots, l$;
- dato un input $\boldsymbol{x}$ si trova l'$\boldsymbol{x_p}$ più vicino possibile, secondo una certa distanza d;
- infine a $\boldsymbol{x}$ viene assegnato lo stesso valore di $\boldsymbol{x_p}$, cioè y_p.

La scelta della distanza dipende dal problema in questione ed è una scelta delicata da non sottovalutare poichè è ciò che davvero determina la classificazione.

Intuitivamente, vogliamo distanze che siano una misura di quanto due oggetti siano simili tra loro.

Esempio 4.3. Facendo riferimento al problema di classificazione delle due specie di punti descritto in (3.1.1), possiamo immaginare di scegliere come nostra distanza quella euclidea. Con questa distanza possiamo immaginare la classificazione in maniera visiva: un diagramma di Voronoi a due colori.

Caratteristiche di questo modello:

- non ci sono errori nella classificazione dei TD;
- è flessibile, nel senso che muovendo un punto del TD il modello può cambiare significativamente (in questo senso quello lineare è più rigido).
- a causa di questa flessibilità è soggetto a rumore: se nei TD sono presenti errori (e.g. outliers), questi portano modifiche rilevanti nel modello.

Un modo per limitare il rumore è quello di fare una media sui k punti più vicini invece che solo sul primo. In questo caso (classificazione binaria) a $\boldsymbol{x}$ si assegna la classe più comune fra i k punti.

Questo è il modello k-NN. Si noti che ora ci possono essere errori sulla classificazione dei TD.

Si può fare una cosa ancora più raffinata introducendo dei pesi inversamente proporzionali alla distanza, secondo il ragionamento che i punti più vicini contano di più di quelli lontani. In ogni caso, scegliere il k giusto determina la bontà del modello ed è un sottile equilibrio tra underfitting e overfitting.

4.1.2 Confronto con il modello lineare

- Non si ottiene un modello come nel caso del fit lineare; tocca memorizzare tutti i TD.
- Si ha una stima locale della funzione di classificazione contro la stima globale data dall'approssimazione lineare.
- Nel k-NN il numero effettivo di parametri che descrive il nostro modello è (a grandi linee) l/k: si intuisce il perchè guardando i casi estremi, se $k = l$ per qualunque punto scelga faccio la media di tutti gli l punti e quindi indipendentemente dal punto iniziale ottengo sempre lo stesso risultato che è descrivibile da un parametro. Se invece $k = 1$, lo spazio viene suddiviso in l regioni dove la funzione di classificazione è costante.

4.1.3 Soluzione ottimale per il problema di classificazione

Consideriamo nuovamente il problema trattato nell'esempio 4.2. Se conoscessimo la distribuzione di probabilità $P(\boldsymbol{x}, y)$ con cui vengono generati i punti test, saremmo in grado di costruire la classificazione ottimale: si otterrebbe assegnando a $\boldsymbol{x}$ la classe y' tale che $P(y'|\boldsymbol{x})$ sia la massima possibile. Questa è nota come Bayes Classifier e il suo error rate si chiama Bayes rate. Il Bayes rate è il minimo raggiungibile una volta nota la distribuzione generatrice.

È interessante notare che k-NN riproduce negli intenti il Bayes Classifier; tuttavia lo fa in maniera imperfetta visto che la probabilità in un punto è calcolata in base a quello che succede in un suo intorno ed è stimata in funzione del TD.

4.1.4 Inductive Bias e limiti del k-NN

Inductive Bias del k-NN: la classificazione è assunta essere simile per oggetti vicini secondo la metrica adottata.

Esaminiamo ora alcuni *limiti* di questo modello:

- il *riscalamento* di alcune variabili per rendere gli input ranges simili, influenza la metrica;
- *dati simbolici* richiedono una metrica ad-hoc (e.g. Hamming distance per stringhe di caratteri);
- il *costo computazionale* maggiore è nella fase di predizione, in cui bisogna trovare il dato nei TD più prossimo al punto test:
  - il tempo cresce col numero di dati memorizzati;
  - si usano proximity search algorithms per ottimizzare (e.g. indicizzando il TD in modo conveniente in base alla metrica);
- c'è un costo in termini di spazio per memorizzare i TD;
- i TD devono essere *sufficientemente densi* nella regione di interesse;
- la richiesta precedente è difficile da soddisfare quando i dati hanno *alta dimensionalità*, cioè quando ci sono tante variabili in gioco. Se d sono le dimensioni, il

volume cresce come L^d (dove L è il range dei dati) e se d è grande servono molti TD per riempire tutto lo spazio. Per risolvere il problema, quando possibile si cerca di eliminare le variabili meno importanti per la classificazione;

- infine, k-NN non assomiglia a qualcosa che impara, ma ricorda più una *look-up table* che si consulta quando si vogliono fare predizioni.

5 RETI NEURALI

5.1 LEZIONE DI GIOVEDÌ 10 OTTOBRE

In questo corso ci occuperemo di *reti neurali artificiali*, che hanno molteplici vantaggi rispetto ad altri modelli di apprendimento automatico:

- imparano da esempi;
- sono adatti sia per problemi di regressione, sia di classificazione;
- possono trattare efficacemente dati incompleti e affetti da rumore;
- approssimano funzioni arbitrarie, non necessariamente lineari.

L'ultimo punto è molto importante ed è formalizzato nel teorema di Cybenko, che vedremo in seguito.

5.1.1 Neuroni artificiali

Definizione 5.1. (Neurone). Un neurone è una unità di *processing* che riceve degli input $\boldsymbol{x} = (x_i)$ da una sorgente esterna e, dopo averli opportunamente pesati con una matrice di pesi $W = (w_{ij})$ (che può mutare nel corso dell'apprendimento), restituisce una famiglia di output o_i, secondo le formule

$$\begin{aligned} \mathrm{net}_i(\boldsymbol{x}) &= (W\boldsymbol{x})_i = \sum_j w_{ij}\, x_j \\ o_i(\boldsymbol{x}) &= f\big(\mathrm{net}_i(\boldsymbol{x})\big). \end{aligned} \tag{5.1}$$

Esempio 5.2. Si possono costruire facilmente dei neuroni in grado di calcolare l'AND o l'OR di (due) valori in input x_1, x_2; entrambi i problemi sono linearmente separabili. Per calcolare lo XOR (problema non linearmente separabile) è invece necessaria una rete a due neuroni che combina il lavoro svolto dai due neuroni in grado di processare l'AND e l'OR.

Quest'idea è alla base delle reti neurali: vedremo che ogni *layer* è in grado di trasformare l'input in una forma più semplice e organizzata e tutto questo avviene in modo automatico durante il processo di apprendimento.

Vedremo che due layers sono sufficienti per modellizzare funzioni arbitrarie (anche se con più layers l'efficienza è maggiore), mentre un solo layer non basta.

5.1.2 Perceptron learning algorithm

Sia dato un problema di classificazione binaria (a valori ± 1). Il *perceptron learning algorithm* procede come segue.

- Si inizializza un vettore dei pesi i pesi $\boldsymbol{w}$ e si sceglie un *learning rate* $0 < \eta < 1$.
- Si itera la seguente procedura fino a che non è soddisfatta una condizione d'arresto (per esempio, quando $\boldsymbol{w}$ si assesta su un certo valore):
  - per ogni istanza (*training pattern*) $(\boldsymbol{x}, d)$ si calcola
  $$\text{out} := \text{sgn}(\boldsymbol{w}^T \boldsymbol{x}) \in \{-1, 1\}; \tag{5.2}$$
  - se out $= d$ allora non si cambiano i pesi W;
  - se out $\neq d$ allora si aggiornano i pesi come segue (*Hebbian learning*):
  $$\boldsymbol{w}_{\textbf{new}} = \boldsymbol{w}_{\textbf{old}} + \eta d \cdot \boldsymbol{x}. \tag{5.3}$$
  In questo modo si ha
  $$\boldsymbol{w}_{\textbf{new}}{}^T \boldsymbol{x} = \boldsymbol{w}_{\textbf{old}}{}^T \boldsymbol{x} + \eta d \|\boldsymbol{x}\|^2, \tag{5.4}$$
  cosicché se, per esempio, $d = 1$, la quantità $\boldsymbol{w}^T \boldsymbol{x}$ aumenta. Alternativamente, la (5.4) si può scrivere come segue (*Error-correction learning*):
  $$\boldsymbol{w}_{\textbf{new}} = \boldsymbol{w}_{\textbf{old}} + \frac{1}{2}\eta(d - \text{out})\boldsymbol{x}. \tag{5.5}$$

Nella prossima lezione mostreremo formalmente che il *perceptron learning algorithm* è sempre in grado di imparare ciò che è in grado di rappresentare. Inoltre se il problema è linearmente separabile, l'algoritmo converge (cioè classifica correttamente tutti i training samples) in un numero finito di passi, a differenza del LMS, che può giungere alla classificazione sbagliata dei training samples anche in caso di lineare separabilità.

5.2 LEZIONE DI MARTEDÌ 15 OTTOBRE

Teorema 5.3. (Perceptron Convergence Theorem). *Se il problema di classificazione binario è separabile, il Perceptron Learning Algorithm converge sempre, nel senso che a un certo punto $\boldsymbol{w}$ si assesta e classifica correttamente i training data.*

Dimostriamo il teorema per un learning rate fisso (nel proseguio scegliamo $\eta = 1$ per semplicità). Denotiamo l'insieme dei training data con $(\boldsymbol{x}_i, d_i)$, quello che mostriamo è che l'algoritmo non può sbagliare a classificare una sequenza di TD (anche ripetuti) più di un certo numero di volte.

Immaginiamo quindi che all'algoritmo venga data in pasto una certa sequenza di TD, questo commetterà un certo numero di errori, indichiamo con $\boldsymbol{x}(n)$ l'n-esimo dato mal classificato; in parallelo otterremo una successione di $\boldsymbol{w}$, con $\boldsymbol{w}(n)$ intendiamo il valore di $\boldsymbol{w}$ aggiornato dall'algortimo dopo l'errore n-esimo. Assumiamo (sempre per semplicità) $\boldsymbol{w}(0) = 0$.

Avendo scelto $\eta = 1$, dopo n errori $\boldsymbol{w}$ sarà una somma di n termini

$$\boldsymbol{w}(n) = \sum_i \boldsymbol{x}_+(j) - \sum_k \boldsymbol{x}_-(k) \tag{5.6}$$

dove $\boldsymbol{x}_+$ e $\boldsymbol{x}_-$ sono rispettivamente quelli per cui $d = +1$ e $d = -1$.

L'idea alla base è quella di vincolare dal basso e dall'alto la norma di $\boldsymbol{w}(n)$ e mostrare che i due vincoli entrano in conflitto quando n il numero di errori è troppo grande.

Sappiamo innanzitutto che il problema è separabile, quindi esiste (più di) una soluzione $\boldsymbol{w}_0$ tale che

- $\boldsymbol{w}_0 \cdot \boldsymbol{x}_+ > 0$ per tutti gli $\boldsymbol{x}_+$ nei TD.
- $\boldsymbol{w}_0 \cdot \boldsymbol{x}_- < 0$ per tutti gli $\boldsymbol{x}_-$ nei TD.

Definiamo preliminarmente

$$\alpha = \min_i \quad |\boldsymbol{w}_0 \cdot \boldsymbol{x}_i| \tag{5.7}$$

$$\beta = \max_i \quad |\boldsymbol{x}_i|^2 \tag{5.8}$$

- Vincolo dal basso:

$$|\boldsymbol{w}_0|^2 |\boldsymbol{w}(n)|^2 \geq (\boldsymbol{w}_0 \cdot \boldsymbol{w}(n))^2 \geq (n\alpha)^2 \tag{5.9}$$

$$\implies |\boldsymbol{w}(n)|^2 \geq (n\alpha)^2 / |\boldsymbol{w}_0|^2 \tag{5.10}$$

- Vincolo dall'alto:

  Nel caso di falso negativo

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{x}_+(n) \tag{5.11}$$

$$\implies |\boldsymbol{w}(n+1)|^2 = |\boldsymbol{w}(n)|^2 + |\boldsymbol{x}_+(n)|^2 + 2\boldsymbol{w}(n) \cdot \boldsymbol{x}_+(n)$$

  dato che $\boldsymbol{x}_+(n)$ è un errore $\boldsymbol{w}(n) \cdot \boldsymbol{x}_+(n) < 0$ e quindi vale

$$|\boldsymbol{w}(n+1)|^2 \leq |\boldsymbol{w}(n)|^2 + |\boldsymbol{x}_+(n)|^2 \leq |\boldsymbol{w}(n)|^2 + \beta \tag{5.12}$$

  Anche nel caso di falso positivo, tenendo presente che $\boldsymbol{w}(n) \cdot \boldsymbol{x}_-(n) > 0$ si riottiene la disuguaglianza (5.12). Abbiamo allora

$$|w(n)|^2 \leq n\beta \tag{5.13}$$

mettendo insieme le due si ha

$$(n\alpha)^2 / |\boldsymbol{w}_0|^2 \leq |w(n)|^2 \leq n\beta \tag{5.14}$$

Il problema è che il limite inferiore cresce quadraticamente in n mentre quello superiore solo linearmente, di conseguenza il vincolo non può valere per n troppo grande. Da questo segue un upper bound sul numero di errori che deve essere inferiore al più grande n_{max} che soddisfa la (5.14) (notare che $\boldsymbol{w}_0$ differenti possono portare a n_{max} diversi).

Se invece il problema di classificazione non è linearmente separabile, l'algoritmo può sviluppare cicli senza convergere.

5.2.1 Differenze tra LMS e Perceptron Learning Algorithm

- In LMS l'errore è trattato in maniera continua $\delta = d - \boldsymbol{x} \cdot \boldsymbol{w}$, nell'algoritmo precedente si cercano di minimizzare i TD classificati male (discreto):

  $(d - \boldsymbol{x} \cdot \boldsymbol{w})$ VS $(d - \text{sgn}(\boldsymbol{x} \cdot \boldsymbol{w}))$.
- per il motivo precedente LMS perseguenedo il suo obiettivo, può sbagliare a classificare i TD;
- LMS a differenza del Perceptron Learning Algorithm converge anche se il problema non è separabile.

5.2.2 Activation Function

Definizione 5.4. (Activation Function). In una rete neurale la funzione di attivazione di un nodo è quella che definisce l'output dati gli input.

Forniamo un qualche esempio di activation function:

- Nell'algoritmo precedente è stata usata la funzione $\text{sgn}(x)$;
- Si può usare come funzione di attivazione anche una funzione che è combinazione lineare degli input;
- L'esempio molto usato di funzione non lineare è la sigmoidal logistic function

$$f_\sigma(x) = \frac{1}{1 + e^{-ax}} \tag{5.15}$$

  al variare del parametro a si possono riprodurre con una certa approssimazione i due casi precedenti;
- la risposta di un nodo agli input può essere anche stocastica, cioè dato un certo input con una data probabilità restituisce valori diversi;
- ReLu (Rectifier Linear Unit)

$$f(x) = \begin{cases} 0, & \text{se } x < 0 \\ x, & \text{se } x > 0 \end{cases} \tag{5.16}$$

  diventa una scelta standard per modelli deep learning come vedremo.

Essendo la sigmoidal function regolare (derivabile) si può fare LMS con gradient descent cercando di minimizzare la quantità

$$E(\boldsymbol{w}) = \sum_{TD} (d_i - out(\boldsymbol{x}_i))^2 \tag{5.17}$$

5.2.3 Neural Networks

Una NN è una rete di unità interconnesse.

Una NN è descritta da:

- Unità: come operano cioè l'activation function;

- architettura: numero di unità, topologia, numero di strati;
- Algoritmo attraverso cui apprende.

Si possono dividere in due tipi

- Feedforward: la direzione delle computazioni è da input ad output, nel senso che ciò che viene elaborato da un certo strato viene mandato al successivo e mai indietro;
- Recurrent NN: In questo caso ci possono essere loop interni.

Esempio 5.5. La rappresentazione (la funzione $h(x)$ in uscita) di una NN con uno solo strato nascosto è matematicamente (se f è la funzione di attivazione):

$$h(x) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right) \tag{5.18}$$

L'interpretazione degli strati interni è che producono una rappresentazione interna degli input e estraggono informazioni di un livello più alto dai dati immessi.

La non linearità è importante, si può mostrare velocemente guardando (5.18) che NN a più strati con funzione di attivazione lineare sono riconducibili ad una NN a strato singolo.

Teorema 5.6. (Universal Approximation). *Una NN con singolo strato nascosto (con sigmoidal logistic function) può approssimare arbitrariamente bene ogi funzione continua (premesso che abbia abbastanza unità nello strato nascosto).*

Sostanzialmente è un teorema di esistenza, non dice quante unità servono per approssimare la funzione con una certa precisione. Anche se in teoria è possibile fare tutto con NN a singolo strato nascosto, le NN a più strati possono avere il vantaggio di essere più veloci da ottimizzare e richiedere meno unità per svolgere lo stesso compito.

Il potere espressivo di una NN è influenzato fortemente dal numero di unità e dall'architettura.

Nelle prossime lezioni vedremo algoritmi per l'apprendimento di una rete neurale e come decidere la struttura più adatta per un certo obiettivo.

5.3 Lezione di giovedì 17 ottobre

Definizione 5.7. Il *loading problem* consiste nel sapere se, dato un insieme di training data, esiste un insieme di pesi tale che la rete sia consistente con tali esempi. Si può mostrare che il loading problem è NP-completo.

Ci occuperemo ora di determinare un algoritmo di apprendimento per una rete neurale.[1]

[1] Fino al 1986, questo problema era considerato troppo difficile per essere risolto.

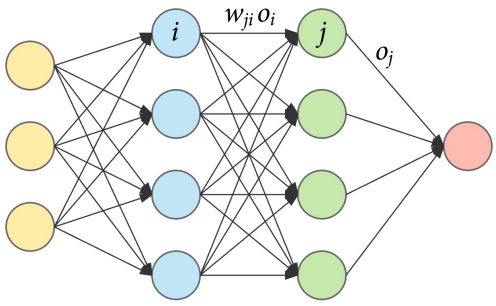

Figura 5.1: Schema di una rete neurale con due *hidden layers* e un solo nodo di output.

5.3.1 Algoritmo di Backpropagation

Vedremo l'algoritmo di *Backpropagation*. Esso sostanzialmente consiste nell'applicare il metodo del gradiente cercando il vettore dei pesi $\boldsymbol{w}$ che minimizzi l'errore E, per ora inteso come l'errore $E = E_p$ relativo a un singolo training example. Per calcolare il gradiente di $\boldsymbol{w}$ occorre saper calcolare le derivate parziali di E rispetto a ciascuno dei pesi w_{ji}. Sia quindi fissato un neurone j, a cui arriva l'input

$$\text{net}_j = \sum_{i \in I} w_{ji} o_i \tag{5.19}$$

dove i nodi in I sono quelli che hanno il nodo j tra i propri nodi in output e o_i è l'output del nodo i. Si ha:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}}. \tag{5.20}$$

Ricordiamo che si ha

$$o_j = f_j(\text{net}_j) \Rightarrow \frac{\partial o_j}{\partial \text{net}_j} = f_j'(\text{net}_j). \tag{5.21}$$

Inoltre dalla (5.19) segue che

$$\frac{\partial \text{net}_j}{\partial w_{ji}} = o_i. \tag{5.22}$$

Quindi gli ultimi due termini dell'RHS della (5.19) sono facilmente determinabili. Si tratta ora di calcolare $\partial E / \partial o_j$. Nel caso in cui j è un nodo di output, cioè $o_j = y_j$, allora il problema non si pone:

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y_j}. \tag{5.23}$$

Nel caso in cui j sia un nodo più interno, occorre procedere ricorsivamente. Sia $L = \{l_1, \ldots, l_k\}$ l'insieme dei nodi che ricevono input direttamente dal nodo j. Allora la variazione della funzione E rispetto a o_j dipende solo dalla variazione della funzione E rispetto a net_l (per i vari $l \in L$):

$$\frac{\partial E}{\partial o_j} = \sum_L \frac{\partial E}{\partial \text{net}_l} \frac{\partial \text{net}_l}{\partial o_j} = \sum_L \frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial \text{net}_l} \frac{\partial \text{net}_l}{\partial o_j} = \sum_L \frac{\partial E}{\partial o_l} f_l'(\text{net}_l) w_{lj}. \tag{5.24}$$

Osserviamo che compaiono ancora termini della forma $\partial E/\partial o_l$, ma questa volta, l è un nodo "un livello più vicino" all'output rispetto a j. Assumendo che la rete neurale in questione non abbia cicli, è possibile determinare iterativamente, a partire dai nodi output, tutti i termini $\partial E/\partial o_l$ e dunque calcolare il gradiente di E.

Spesso assumiamo come errore (relativo al pattern p) consideriamo

$$E_p = \frac{1}{2}\sum_r (d_{pr} - y_r)^2, \tag{5.25}$$

dove il vettore $\boldsymbol{d_p}$ è il vettore di output del pattern p, le cui componenti sono indicizzate da r. In tal caso si ha

$$\frac{\partial E}{\partial y_j} = d_{pj} - y_j. \tag{5.26}$$

Abbiamo ora tutti gli ingredienti per l'algoritmo. Dopo aver inizializzato (a caso) il vettore dei pesi $\boldsymbol{w}$, un generico passo dell'algoritmo consiste nel muovere il vettore $\boldsymbol{w}$ nella direzione opposta a quella del gradiente della funzione E_{tot}, dove E_{tot} è l'errore totale, somma (o media) degli errori su tutti i training samples:

$$\boldsymbol{w}_{\mathbf{new}} = \boldsymbol{w}_{\mathbf{old}} + \eta\Delta\boldsymbol{w} \tag{5.27}$$

con

$$\Delta\boldsymbol{w} = -\nabla E_{\text{tot}} = -\nabla\sum_p E_p = -\sum_p \nabla E_p = \sum_p \Delta^{(p)}\boldsymbol{w} \tag{5.28}$$

dove $\Delta_p\boldsymbol{w} = -\nabla E_p$ è la variazione di $\boldsymbol{w}$ dovuta al pattern p. In componenti, si ha

$$\Delta\boldsymbol{w}_{ji} = \sum_p \Delta_p w_{ji} = \sum_p \left(-\frac{\partial E_p}{\partial w_{ij}}\right) = \sum_p \delta_{pj} o_i, \tag{5.29}$$

dove, per quanto visto, si ha

$$\delta_j^{(p)} = \begin{cases} (d_{pj} - y_j) f_j'(\text{net}_j) & \text{se } j \text{ è un nodo di output} \\ \sum_L \delta_{pl} w_{lj} \cdot f_j'(\text{net}_j) & \text{se } j \text{ è un nodo interno.} \end{cases} \tag{5.30}$$

5.4 Lezione di venerdì 18 ottobre

5.4.1 Euristiche per Backpropagation

Il training di una Neural Network (attraverso l'algoritmo di Backpropagation) non è indolore. Vanno operate diverse *scelte*, necessarie per evitare vari problemi, tra cui l'overfitting e la discesa verso minimi locali "cattivi" (ricordiamo che il problema di minimizzazione è in generale non convesso).

I pesi iniziali

È preferibile scegliere come pesi iniziali dei valori randomici vicini a zero. È però importante *evitare*:

- valori tutti nulli,
- valori grandi in modulo,
- valori tutti uguali.

Una possibilità è fissare un range (e.g. $[-0.7, 0.7]$) e distribuire i pesi con probabilità uniforme. Un'altra è fissare un valore $a > 0$ (per esempio 0.7) e distribuire i pesi relativi a un nodo i nel range $[-2a/\text{fan-in}_i, 2a/\text{fan-in}_i]$.

Definizione 5.8. (Fan-in). Definiamo *fan-in* il numero di input di una unità di apprendimento.

Attenzione, però! Quest'ultima scelta non è buona quando il fan-in è troppo alto e, in ogni caso, non va mai fatta per le unità di output.

Mancanza di convessità

La mancanza di convessità non è un enorme problema: di solito un "buon" punto di minimo è sufficiente.

Si osservi, tra l'altro, che tante volte non cerchiamo un vero e proprio minimo, perché questo potrebbe anche essere sintomo di overfitting (minimizziamo R_{emp}, non R). C'è poi il problema dell'overtraining.

Definizione 5.9. (Overtraining). Il training di una NN è in grado di incrementare la dimensione dello spazio delle ipotesi durante il training, e quindi il training error scende a 0 non perché raggiungiamo un minimo globale, ma perché la rete neurale diventa troppo complessa (stiamo "creando" nuovi minimi). Questo è l'*overtraining*.

Per trovare questo "buon" punto di minimo è importante sempre eseguire l'algoritmo più volte con diverse configurazioni iniziali randomiche. Alla fine si può calcolare la varianza dell'errore per valutare il modello e, per dare un risultato, si può scegliere:

- il punto con l'errore minimo,
- il punto con l'errore mediano,
- la "media" dei vari punti ottenuti.

Batch o online?

L'apprendamento può essere eseguito conoscendo a priori tutto il training set (Batch) o minimizzando l'errore prendendo un input per volta (stochastic/online). Il Batch è più accurato e stabile ma più lento. Il metodo online può aiutare ad evitare i minimi locali (effetto regolarizzante; aggiunge "rumore" al processo di apprendimento).

si fissano $\boldsymbol{w}$ e η;
while *non converge* **do**
  $\Delta \boldsymbol{w} \leftarrow -\nabla E_{\text{tot}} = -\sum_p \nabla E_p$;
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \Delta \boldsymbol{w}$;
end

Algoritmo 1: Metodo Batch

si fissano $\boldsymbol{w}$ e η;
while *non converge* **do**
  foreach *pattern p (in ordine random)* **do**
    $\Delta_p \boldsymbol{w} \longleftarrow = -\nabla E_p$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \Delta_p \boldsymbol{w}$;
  end
end

Algoritmo 2: Metodo online

Il minibatch SGD è una vita di mezzo tra il metodo batch e il metodo online, in cui considero un blocco di *mb* esempi per volta. È importante che tali blocchi siano scelti in maniera randomica, e che siano diversi di epoca in epoca (lo stesso principio vale per il metodo online).

Il minibatch è particolarmente utile quando il training set è molto grande e/o non viene letto tutto insieme. In tali casi è possibile anche avere una sola epoca.

Velocità di apprendimento

Valori alti di η rendono l'algoritmo veloce ma instabile, mentre valori piccoli lo rendono stabile ma lento. Non solo, ma valori troppo piccoli possono far sì che si rimanga bloccati attorno ad un minimo locale con alto training error.

La scelta di η deve essere quindi bilanciata col metodo che si usa: in generale, col Batch si useranno valori *più grandi* che con l'online.

Un metodo per valutare la qualità del valore di η utilizzato può essere quella di plottare e guardare la *curva di apprendimento*.

Definizione 5.10. (Curva di apprendimento). La *curva di apprendimento* o *learning curve* è il grafico che ha in ordinate il training error e in ascisse il numero di epoche.

Con i metodi online e minibatch, spesso il gradiente non tende a zero in corrispondenza di un minimo. È pertanto opportuno far *decrescere* il valore di η iterazione dopo iterazione. Più precisamente si possono fissare

- un valore iniziale η_0 (da determinarsi per tentativi),
- un numero di iterazioni $\tau \approx 100 \sim 300$,
- un valore finale $\eta_\tau \approx \eta_0/100$

e far decrescere linearmente η da η_0 a η_τ nelle prime τ iterazioni (poi lo si lascia fisso a η_τ).

Parliamo ora del *momentum*. Si tratta di un sistema per smorzare le oscillazioni e le brusche frenate e accelerazioni (proprio come in fisica). Fissato $\alpha \in [0, 1)$, implementare il momentum vuol dire usare le formule

$$\begin{aligned} \Delta \boldsymbol{w}_{\text{new}} &= -\eta \frac{\partial E}{\partial \boldsymbol{w}}(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}_{\text{old}} \\ \boldsymbol{w}_{\text{new}} &= \boldsymbol{w}_{\text{old}} + \Delta \boldsymbol{w}_{\text{new}}. \end{aligned} \tag{5.31}$$

Il momentum così come appena presentato è usato solitamente col metodo Batch. Coi metodi online e minibatch non sempre viene usato, e, eventualmente, si implementa smussando il gradiente con la *media* dei gradienti precedenti.

C'è anche una variante, detta *Nesterov momentum*, che risulta essere migliore per il metodo Batch ed è descritta dalle formule

$$\begin{aligned} \underline{\boldsymbol{w}} &= \boldsymbol{w}_{\text{old}} + \alpha \Delta \boldsymbol{w}_{\text{old}} \\ \Delta \boldsymbol{w}_{\text{new}} &= -\eta \frac{\partial E}{\partial \boldsymbol{w}}(\underline{\boldsymbol{w}}) + \alpha \Delta \boldsymbol{w}_{\text{old}} \\ \boldsymbol{w}_{\text{new}} &= \boldsymbol{w}_{\text{old}} + \Delta \boldsymbol{w}_{\text{new}}. \end{aligned} \tag{5.32}$$

Altre possibilità ancora per modificare la velocità di apprendimento sono:

- usare η diversi per ciascun parametro (quello che era un valore scalare nelle formule precedenti diventa una matrice diagonale);
- tecniche di adattamento automatico di η durante il training;
- usare metodi del secondo ordine, come il metodo di Newton, i metodi di Newton approssimati, l'algoritmo del gradiente coniugato. Il metodo di Newton (e altri che calcolano esplicitamente la matrice Hessiana) possono però presentare numerosi problemi legati al tempo di computazione elevato e alla presenza di molti punti di sella nello spazio dei pesi (essi crescono esponenzialmente in spazi di grandi dimensioni).

Criteri di arresto

Ci sono molti modi per decidere il criterio di arresto del Backpropagation. Tra questi:

- l'errore E_{tot} è più piccolo di una certa tolleranza;
- il massimo degli errori $\max_p E_p$ è più piccolo di una certa tolleranza;
- non ci sono più variazioni significative nei pesi o nell'errore (meno del 0.1%).

In ogni caso non bisogna far passare un numero eccessivo di epoche.

Regolarizzazione

Per evitare l'overfitting bisogna aggiungere della regolarizzazione. Questo può avvenire direttamente, attraverso la *penalizzazione*, o indirettamente, fermando presto la computazione (*early stopping*, in questo caso si deve usare il validation set, e se ne parlerà in futuro).

Concentriamoci sulla penalizzazione (cui a volte ci si riferisce direttamente col termine "regolarizzazione"). Si tratta di prendere un parametro $\lambda > 0$ molto piccolo (nel caso Batch, $\lambda \approx 0.01$) nella fase di model selection e aggiungere il termine $\lambda \|\boldsymbol{w}\|^2$ a $E_{\text{tot}}(\boldsymbol{w})$. Stiamo di fatto aggiungendo $-2\lambda \boldsymbol{w}_{\text{old}}$ al LHS della formula

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} + \eta \Delta \boldsymbol{w}_{\text{old}}. \tag{5.33}$$

Osservazione 5.11. Si osservi che *non* stiamo moltiplicando η e λ fra loro.

Osservazione 5.12. Chiameremo con *Loss* la funzione da minimizzare (con la penalizzazione) e con *errore* e *rischio* la somma $\sum_p \left(d_p - o(\boldsymbol{x_p})\right)^2$.

Osservazione 5.13. Se nel caso Batch avremmo usato un certo valore di penalizzazione λ, per essere coerenti, nel caso online/minibatch dovremo scegliere un parametro molto più piccolo, dell'ordine di $\lambda \cdot mb/(\#\text{ total patterns})$.

6 VALIDATION

6.1 LEZIONE DI GIOVEDÌ 24 OTTOBRE

6.1.1 Consigli per il progetto - benchmark

È difficile capire la correttezza di una rete neurale dai risultati perché spesso una rete funziona anche con piccoli errori di implementazione.

MONK TEST Un primo test si può fare sul MONK data set (risultati da riportare nel project report), che consiste in 3 task di classificazione binaria, dove è facile ottenere alta accuratezza.

Una rete neurale corretta implica buoni risultati nel Monk test, non viceversa.

- Per fare il test bastano 2-5 dati, però se se ne usano così pochi la rete è sensibile ai pesi iniziali.
- Il risultato consiste di un output binario. Si consiglia di usare tanh come activation function.
- Se ho 17 input mi servono 17 neuroni al primo layer. 1-hot encoding (un neurone con 17 valori possibili) sbaglia perché il risultato dipende dall'ordinamento/coding degli input.

6.1.2 Model Assessment and Model Selection

La validation ha due obiettivi:

MODEL SELECTION: stimare la performance di diversi modelli per scegliere il migliore, bilanciando l'accuracy sui training data e la model complexity.

Formalmente, dato un parametro θ che varia la complessità del modello, vogliamo trovare il valore di θ che minimizza l'errore sui 'test' data (validation data).

MODEL ASSESSMENT: scelto il modello, stimare l'efficienza (the expected error) della mia rete.

Notiamo che i training set non sono un buon estimatore dell'errore (non si capisce solo dai TR data quando si è in under-fitting o in over-fitting), bensì possiamo dare una stima della bontà del modello analiticamente (VC-dimension) o attraverso ricampionamento (stima diretta dell'errore su un altro sample non usato per training o model selection).

I metodi di ricampionamento includono:

- cross-validation (hold-out, K-fold cross-validation)
- bootstrap

È importante non confondere model selection con model assessment, infatti se per scegliere il modello si usassero i test data, si sovrastimerebbe l'accuracy.

GOLD RULE: **Keep separation between goals and use separate data sets**

HOLD-OUT Il metodo dell'hold-out consiste nel dividere i i dati in tre insiemi disgiunti: Training (TR), Validation (VA) e Test (TS).

Il primo serve per fare il fit del modello, il secondo per scegliere il miglior modello (scelta degli hyper-parametri) e l'ultimi per stimare le performance (model assessment). (TR, VA, TS) $\sim (0.5, 0.25, 0.25)$

L'hold-out funziona se abbiamo abbastanza dati (quanti dipende da model complexity, signal to noise ratio, …)

K-FOLD CV Partizioniamo l'insieme dei dati iniziali D in k sottoinsiemi disgiunti D_k. Facciamo il traning della nostra rete su D/D_i e testiamo su D_i. Abbiamo K modelli (vedremo in seguito come sceglierne uno) e K stime dell'efficienza su cui possiamo mediare.

Problema: computazionalmente molto costoso

ESEMPIO DI MODEL SELECTION E MODEL ASSESSMENT

- Dividiamo i dati in TR e test set (hold-out o CV).
- Model selection - Sul TR set, usiamo K-fold CV, ottenendo K partizioni $(TR^*, VL)_{i=1}^{k}$. Su ciascuna partizione troviamo i migliori hyper-parametri del modello (e.g. ordine polinomiale, lambda of ridge regression, numero di unità nella rete neurale...). Facciamo una grid-search con molto possibili valori degli hyper-parametri.
- Fare il training su TR per il modello finale
- Model assessment - fare l'evaluation sul test set

GRID SEARCH Supponiamo di aver h hyper-parametri, per trovare i valori migliori per la nostra rete, creaimo una tabella con h dimensionale, dove in ogni asse ci sono alcuni valori dell'hyper-parametro i-esimo.

Exhaustive Grid Search: faccio il model assessment per ogni celletta. Tuttavia l'efficienza scala male (prodotto cartesiano degli insiemi dei valori degli hyper-parametri). Quindi può essere meglio muoversi lungo alcune righe per fissare alcuni hyper-parametri indipendentemente da altri (più veloce, ma meno preciso). Alternativamente si può scorrere sulla griglia a grandi intervalli e poi fare una ricerca più fine su una sotto-griglia che ha ottenuto buoni risultati.

Possiamo evitare di essere sensibili ad un particolare partizionamento degli esempi?

- Stratificazione: raggruppare campioni della popolazioni in gruppi omogenei prima del resample.
- Classificazione: ogni classe è rappresentata con le stesse proporzioni come nell'intero data set.
- Folder composition: controlla se l'ordine nei dati ha un significato specifico.

- Ripetere hold-out / CV: la media ci fornisce un risultato globale, particolarmente utile per comparare diversi modelli.

Con pochi dati è difficile dire se un sample è significativo o no.

6.2 LEZIONE DI VENERDÌ 25 OTTOBRE

Il nostro obiettivo è

- ottenere il modello migliore dati i training data (model selection)
- fare una stima di quanto è buono questo modello (model assessment)

Per entrambi questi task possiamo usare diversi metodi:

- simple validation
- cross validation (k-fold)

6.2.1 Model Selection

Sia $f \in \mathcal{F}$ una funzione che mappa gli input della rete agli output e sia $L : \mathcal{Z} \times \mathcal{F}$ la loss function. Definiamo il rischio di un certo modello, $R(f) = E_Z[L(z, f)] = \int_Z L(z, f)p(z)$, dove z è ad esempio la coppia (x, y), con x i valori in input e y l'output corretto.

Purtroppo non conosciamo $p(z)$, e non conosciamo tutte le $L(z, f)$, al variare di z (conosciamo solo un numero finito di coppie (x, y)). Useremo pertanto il rischio empirico $\hat{R}(f, D_n) = \frac{1}{n}\sum_{i=1}^{n} L(z_i, f)$ con D_n l'insieme dei dati (delle coppie (x, y)).

SIMPLE VALIDATION Data una famiglia di modelli con hyper-parametro θ, partizioniamo il training set D_n in 2 parti (D^{tr} e D^{va}). Per ogni valore θ_m dell'hyper-parameter θ:

- selezioniamo il modello che minimizza l'errore sui training data

$$f^{\star}_{\theta_m} = \arg\min_{f \in \mathcal{F}_{\theta_m}} \hat{R}(f, D^{tr})$$

- stimiamo l'errore di questo modello sui validation data

$$R(f^{\star}_{\theta_m}) \sim \hat{R}(f^{\star}_{\theta_m}, D^{va}) = \frac{1}{va}\sum_{z_i \in D^{va}} L(z_i, f^{\star}_{\theta_m}(D^{tr}))$$

Selezioniamo dunque il valore dell'hyper parametro che minimizza la nostra stima dell'errore

$$\theta^{\star}_m = \arg\min_{\theta_m} R(f^{\star}_{\theta_m})$$

Scegliamo infine il modello che minimizza l'errore su tutti i dati (TS+VA), con il constrain che il modello viene scelto dall'insieme delle funzioni con $\theta = \theta_m$.

$$f^{\star}(D_n) = \arg\min_{f \in \mathcal{F}_{\theta^{\star}_m}} \hat{R}(f, D_n)$$

CROSS-VALIDATION Data una famiglia di modelli con hyper-parametro θ, partizioniamo il training set D in K parti uguali $(D^1, D^2, \ldots, D^k)$. Per ogni valore θ_m dell'hyper-parameter θ:

- per ogni sottoinsieme D^k, chiamato $\bar{D}^k$ il suo complementare:
  - selezioniamo il modello che minimizza l'errore su $\bar{D}^k$ (training della rete)

$$f^{\star}_{\theta_m}(\bar{D}^k) = \arg\min_{f \in \mathcal{F}_{\theta_m}} \hat{R}(f, \bar{D}^k)$$

  - stimiamo l'errore di questo modello sui D_k

$$R(f^{\star}_{\theta_m}(\bar{D}^k)) \sim \hat{R}(f^{\star}_{\theta_m}(\bar{D}^k), D.2^k) = \frac{1}{|D^k|} \sum_{z_i \in D^k} L(z_i, f^{\star}_{\theta_m}(\bar{D}^k))$$

- stimiamo l'errore medio del modello con la media sugli errori, al variare di D_K

$$R(f^{\star}_{\theta_m}(D_n)) \sim \frac{1}{K} \sum_{k} R(f^{\star}_{\theta_m}(\bar{D}^k))$$

Selezioniamo quindi il valore dell'hyper parametro che minimizza la nostra stima dell'errore.

$$\theta^{\star}_m = \arg\min_{\theta_m} R(f^{\star}_{\theta_m}(D_n))$$

Scegliamo infine il modello che minimizza l'errore su tutti i dati, con il vincolo che il modello viene scelto dall'insieme delle funzioni con $\theta = \theta_m$.

$$f^{\star}(D_n) = \arg\min_{f \in \mathcal{F}_{\theta^{\star}_m}} \hat{R}(f, D_n)$$

6.2.2 Estimation of the Risk

SIMPLE VALIDATION

- dividiamo il data set D_n in 2 parti (D^{tr}, D^{te})
- selezioniamo il modello che minimizza l'errore sui training data

$$f^{\star}(D^{tr}) = \arg\min_{f \in \mathcal{F}} \hat{R}(f, D^{tr})$$

- stimiamo l'errore del modello come media sui test data dell'errore degli stessi sul modello

$$R(f^{\star}(D^{tr})) \sim \hat{R}(f^{\star}(D^{tr}, D^{te})) = \frac{1}{te} \sum_{z_i \in D^{te}} L(z_i, f^{\star}(D^{tr}))$$

CROSS – VALIDATION Dividiamo il nostro data set in K subset. Per ogni D^k facciamo il training del nostro modello su $\bar{D}^k$ e stimiamo l'errore come media della loss function calcolata su D^k.

$$f^\star(\hat{D}^k) = \arg\min_{f \in \mathcal{F}} \hat{R}(f, \hat{D}^k)$$

$$R(f^\star(\bar{D}^k)) \sim \hat{R}(f^\star(\bar{D}^k, D^k)) = \frac{1}{|D^k|} \sum_{z_i \in D^k} L(z_i, f^\star(\bar{D}^k))$$

L'errore restituito è la media dell'errore stimato sui vari D^k

$$R(f^\star(D_n)) \sim \frac{1}{K} \sum_k R(f^\star(\bar{D}^k))$$

[leave-one-out] se $k = n$, si testa solo su un dato.

MODEL ASSESSMENT E MODEL SELECTION In generale vorremmo fare sia model selection che model assessment, allora dobbiamo unire i metodi sopra presentati. Ad esempio:

1. dividiamo il data set in 3 parti (tr, va, te).
2. dividiamo i dati in 2 insiemi e applichiamo la cross-validation sul primo insieme per model selection e usiamo il secondo per fare il test.
3. double cross-validation: per ogni subset, facciamo una altro cross-validation con gli altri K-1 altri subset

DOUBLE CROSS VALIDATION Selezioniamo una famiglia di funzioni con hyper-paramentro θ. Diviiamo il training set D in K sottoinsiemi. Per ogni D^k:

1. selezioniamo il miglior modello $f^\star(\bar{D}^k)$ facendo cross-validation su $\bar{D}^k$
2. stimiamo $R(f^\star(\bar{D}^k)) \sim \hat{R}(f^\star(\bar{D}^k, D^k)) = \frac{1}{|D^k|} \sum_{z_i \in D^k} L(z_i, f^\star(\bar{D}^k))$

Stimiamo $R(f^\star(D_n)) \sim \frac{1}{K} \sum_k R(f^\star(\bar{D}^k))$

Così otteniamo una stima del rischio, non un modello.

6.2.3 FAQs

Early stopping (decidere quando fermare il training della rete) fa parte della model selection, non bisogna usare test set!

Alternativamente a early stopping per decidere quando fermarsi si possono usare modelli regolarizzati (con stopping criteria).

RANDOM INIT Anche scegliere i pesi iniziali fa' parte della model selection. Come scegliamo i pesi iniziali per il nostro modello finale dati i pesi iniziali dei vari modelli che abbiamo provato?

- si può calcolare la media e la varianza degli errori per diverse prove, alta varianza non va bene.

- Fare la media dei vari pesi iniziali
- Sceglierne uno (model selection)
  - il migliore
  - il mediano
  - random

7 SLT

7.1 LEZIONE DI MARTEDÌ 29 OTTOBRE

In questa lezione ci occupiamo principalmente di statistical learning theory.

7.1.1 Approximating the estimation step

Possiamo usare alcuni approcci per stimare l'errore sul set di validazione per la model selection o sul test set per l'assessment del modello. Ci sono approcci più analitici e altri basati sul resampling. In particulare ci occupiamo di minimizzazione del rischio strutturale, che si basa fortemente sul concentto già incontrato di VC-dimension, che è sostanzialmente una misura della flessibilità descrittiva data dall'insieme delle ipotesi H. Ciò che importa è la capacità dell'insieme delle ipotesi di descrivere bene le configurazioni dello spazio delle istanze, e in particolare vogliamo quindi trovare il massimo numero di punti che può essere correttamente classificato per ogni labeling. Non siamo quindi interessati alla cardinalità $|H|$ dello spazio delle ipotesi.

Definizione 7.1. (Dicotomia). Una dicotomia dello spazio delle istanze X è una partizione dei punti di X in due insiemi (ad esempio tramite labeling con $+1$ o -1). Se $|X| = N$ ci sono 2^N dicotomie.

Definizione 7.2. (Dicotomia rappresentata). Una dicotomia è rappresentata in H se esiste $h \in H$ che realizza quella particolare dicotomia (ovvero divide X esattamente nei due gruppi individuati dalla dicotomia).

Definizione 7.3. (Shattering). H frammenta (shatters) $X \iff$ tutte le possibili dicotomie su X sono rappresentate in H.

Definizione 7.4. (VC-dimension). La VC-dimension di una classe di funzioni H è la massima cardinalità di un insieme di punti X che può essere frammentato da H. La VC-dimension è infinita se $\forall n \in \mathbb{N}, \exists S$ t.c. $|S| = n$ e S è frammentato da H. Quindi in realtà potremmo dire che la VC-dimension è il $\sup\{|X| : X \in \mathcal{X}\}$ con $\mathcal{X}$ insieme degli insiemi shatterati da H.

Notare che alcuni sottoinsiemi di X con cardinalità minore o uguale della VC-dimension potrebbero non essere frammentati da H. Ad esempio tre punti in linea sul piano non sono shatterabili dalle rette, perché esistono due configurazioni (quelle in cui il punto in mezzo ha labeling diverso dai due agli estremi) che non sono generabili con una retta. Però esiste almeno una configurazione in cui tre punti sono divisibili da una retta.

Come esempio generico consideriamo gli iperpiani lineari su uno spazio n-dimensionale. L'upper bound della sua VC-dimension è $n+1$. Però possiamo dare restrizioni su H per ridurre la sua VC-dimension.

ESERCIZI Trovare la VC-dimension di:

- Rette in $\mathbb{R}$ (2).
- Intervalli su $\mathbb{R}$ (2).
- Cerchi entrati nell'origine in 2D (1 e 2), liberi (raggio e centro) (3), raggio fissato (3) e centro libero (3).

NUMERO DI PARAMETRI È correlato ma non è la stessa cosa della VC-dimension, perché potremmo aggiungere parametri ridondanti ma liberi. Inoltre ci possono essere modelli con pochi parametri (anche uno) con VC-dimension infinita (vedi Haykin/Burges). La VC-dim è infinita anche per il nearest-neighbor.

7.1.2 Analytical bound on R

$$R[h] \leq R_{emp}[h] + \epsilon(VC, N, \delta) \tag{7.1}$$

Questa espressione, che abbiamo già visto (**??**), ci dice che il rischio effettivo è al più la somma del rischio empirico sul test set e di un termine detto VC-confidence che dipende da N numero di dati, VC-dim e δ, con probabilità di almeno $1 - \delta$. In generale ϵ diminuisce aumentando N ma aumenta con la VC-dim. Visto che N lo sappiamo, δ lo scegliamo noi, e dobbiamo quindi decidere la VC-dimension, possiamo ottenere un bound sulla parte di rischio in uno scenario negativo (perché vale con probabilità $1 - \delta$ - se $N > VC$ e quindi in generale sarà molto meno di così). Quindi abbiamo un modo per stimare l'errore su dataset futuri dalla VC-dim e dall'errore di training, che è una cosa molto utile per scegliere un modello alla descrizione dei nostri dati.

Per molte classi di ipotesi sensate (e.g. quelle lineari) la VC-dimension è lineare nel numero di parametri. Questo ci mostra che per imparare bene ci serve un numero di esempi lineare nella VC-dimension (e quindi nel numero di parametri).

7.1.3 Structurak risk minimization

Usa la VC-dimension come parametro di controllo per minimizzare il bound di generalizzazione su R. Se ci limitiamo a VC-dimension finite, possiamo definire una struttura annidata dei set di ipotesi (di un certo tipo) in accordo con la loro VC-dimension. Per esempio:

- NN con aumento di unità nascoste (roughly),
- polinomi di grado crescente,
- valori crescenti di c, dove per regolarizzare imponiamo $\|w\| < c$,
- aumento del numero di nodi in un decision tree (roughly).

All'aumentare della VC-dimension diminuisce l'errore empirico ma aumenta la VC-confidence e il ruolo della SRM è di trovare un buon trade-off tra i due termini. Quindi è un processo di *model selection*, perché lo usiamo per scegliere il modello con il miglior bound sul rischio sui dati nuovi. Bisogna fare attenzione: il termine di

VC-confidence può diventare molto grande molto in fretta, molto più di quella che è l'entità dell'effetto dell'overfitting.

Il bound è molto importante perché è indipendente dai dettagli di modelli specifici e dagli algoritmi di apprendimento, e aiuta a capire il ruolo importante del controllo della complessità e a scegliere in modo ottimale la complessità del modello, e quindi indirizza a una direzione più basata nella teoria e meno empirica (ora molto basata su prove ed errori) nello sviluppo dei modelli. Al momento la stima degli errori predittivi non è molto usata per vari motivi: può essere un bound molto largo e troppo pessimistico per essere affidabile, o perché calcolare la VC-dim di alcune classi di H è difficile. La cosa bella è che ci dà una upper bound probabilistico su classi di funzioni quindi possiamo cercare i min nella classe. Dei bound migliori (più stretti) sono in sviluppo.

Diamo due esempi di approcci pratici. Il primo consiste nello scegliere un modello con struttura e complessità appropriate, fissare il modello (e quindi la VC confidence), e poi minimizzare l'errore di training. In questo approccio possono spesso esserci delle implementazioni implicite dell'SRM ad esempio con regolarizzazioni varie (e.g. Tikhonov, che aggiunge un termine svantaggioso dovuto alla complesità). L'altro esempio fa il contrario: prima fissa il TR error, e poi punta a minimizzare la VC-confidence.

8 SUPPORT VECTOR MACHINES

8.1 LEZIONE DI GIOVEDÌ 31 OTTOBRE

Ci occuperemo ora delle Support Vector Machines (SVM), ovvero algoritmi per risolvere problemi di classificazione binaria con alcune peculiarità.

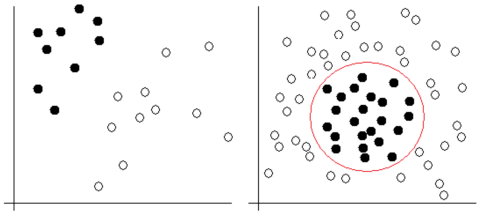

Figura 8.1: Dati linearmente separabili (a sinistra) e non linearmente separabili (a destra).

Assumeremo inizialmente che:

- i dati siano linearmente separabili, come nell'esempio nella figura 8.1 (a sinistra);
- i dati non siano affetti da errore.

Sia quindi dato un training set di dati $\{(\boldsymbol{x_i}, d_i)\}$ con $i = 1, \dots, N$ e $d_i = \pm 1$. Si cerca un una funzione affine $g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ tale che $g(\boldsymbol{x_i}) \geq 0 \iff d_i = +1$.

8.1.1 Margine di separazione e vettori di supporto

Definizione 8.1. (Margine di separazione e vettori di supporto). Il *margine di separazione* ρ di un iperpiano di separazione è la larghezza massima di una striscia centrata nell'iperpiano che non contiene nessun punto $\boldsymbol{x_i}$ (se non nel bordo). In altre parole,

$$\rho = 2 \min \left\{ r(\boldsymbol{x_i}) : i = 1, \dots, N \right\}, \tag{8.1}$$

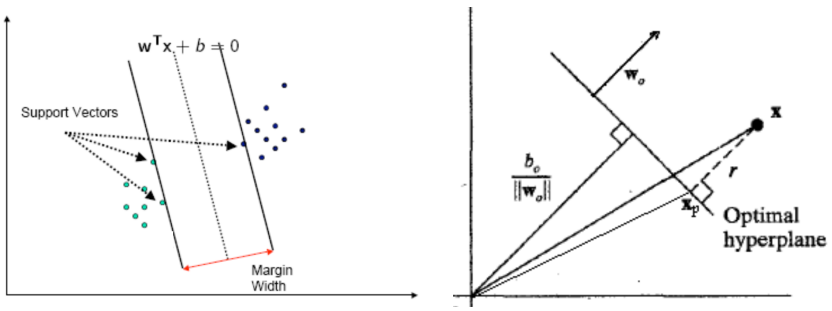

Figura 8.2: Margine di separazione e vettori di supporto.

dove $r(\boldsymbol{x})$ è la distanza di $\boldsymbol{x}$ dall'iperpiano. I *vettori di supporto* sono i punti $\boldsymbol{x_i}$ che minimizzano la distanza r dall'iperpiano. Si veda la figura 8.2, a sinistra.

Sia ora $\boldsymbol{x}$ uno dei punti del training set e sia $\boldsymbol{x_p}$ la sua proiezione sull'iperpiano, in modo che

$$\boldsymbol{x} = \boldsymbol{x_p} + r(\boldsymbol{x})\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \tag{8.2}$$

(si veda la figura 8.2 a destra). Naturalmente si ha:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = r\|\boldsymbol{w}\|. \tag{8.3}$$

Quindi, dato un vettore di supporto $\boldsymbol{x_s}$ il margine di separazione si ottiene così:

$$\rho = 2\,|r(\boldsymbol{x_s})| = 2\,\frac{|g(\boldsymbol{x_s})|}{\|\boldsymbol{w}\|}. \tag{8.4}$$

Osservazione 8.2. Se il margine di separazione non è nullo, possiamo assumere, a meno di rinormalizzare i coefficienti, che se $\boldsymbol{x_s}$ è un vettore di supporto, allora $|g(\boldsymbol{x_s})| = 1$. Dunque g è un iperpiano di separazione se e solo se vale:

$$\begin{cases} g(\boldsymbol{x_i}) \geq 1 & \text{se } b_i = +1 \\ g(\boldsymbol{x_i}) \leq -1 & \text{se } b_i = -1 \end{cases} \tag{8.5}$$

o, equivalentemente:

$$d_i\, g(\boldsymbol{x_i}) = d_i\left(\boldsymbol{w}^T\boldsymbol{x_i} + b\right) \geq 1, \quad i = 1, \ldots, N. \tag{8.6}$$

In tal caso si ha:

$$\rho = \frac{2}{\|\boldsymbol{w}\|}. \tag{8.7}$$

8.1.2 Il problema SVM

Siamo pronti per definire il problema SVM.

Definizione 8.3. (Hard Margin SVM). Dati i training samples $\{(\boldsymbol{x_i}, d_i)\}$ con $i = 1, \ldots, N$ e $d_i = \pm 1$, si cerca l'iperpiano di separazione (normalizzato, vedi l'osservazione 8.2) che massimizzi il margine di separazione (o, equivalentemente, che minimizzi $\|\boldsymbol{w}\|$), ovvero che risolva il problema di minimo

$$\min\left\{\psi(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} : \; d_i\left(\boldsymbol{w}^T\boldsymbol{x_i} + b\right) \geq 1, \; i = 1, \ldots, N\right\}. \tag{8.8}$$

Osservazione 8.4. Osserviamo che la funzione obiettivo ψ è convessa e che i vincoli sono lineari in $\boldsymbol{w}$.

Per risolvere il problema SVM utilizziamo il seguente teorema.

Teorema 8.5. (Karush–Kuhn–Tucker, 1939). *Consideriamo il problema di ottimizzazione*

$$(P) \quad \min\left\{\psi(\boldsymbol{w}) : \; g_i(\boldsymbol{w}) \geq 0, \; i = 1, \ldots, N\right\} \tag{8.9}$$

con ψ e g_i di classe C^1. Si ponga

$$J(\boldsymbol{w}, \boldsymbol{\alpha}) = \psi(\boldsymbol{w}) - \sum_{i=1}^{N} \alpha_i g_i(\boldsymbol{w}) \tag{8.10}$$

Allora, se $\boldsymbol{w_O}$ ($\boldsymbol{w}$ ottimale) è un punto di minimo, valgono le seguenti condizioni:

- $\nabla_w J(\boldsymbol{w_O}) = 0$ *(annullamento della lagrangiana),*
- $g_i(\boldsymbol{w_O}) \geq 0$ *(rispetto dei vincoli),*
- $\alpha_i \geq 0$ *(moltiplicatori positivi),*
- $\alpha_i g_i(\boldsymbol{w_O}) = 0$ *(i moltiplicatori dei vincoli inattivi sono nulli).*

Inoltre, se $\boldsymbol{w_O}$ è un punto stazionario per J e valgono oppurtune ipotesi su ψ e g_i (per esempio ψ convessa e g_i lineari), le condizioni implicano che $\boldsymbol{w_O}$ è un punto di minimo.

Nel nostro caso, consideriamo il vettore $\tilde{\boldsymbol{w}} = (\boldsymbol{w}, b)$ come incognita e $\tilde{\boldsymbol{w}}_{\boldsymbol{O}} = (\boldsymbol{w_O}, b_O)$ il punto di minimo. La condizione di annullamento della lagrangiana diventa

$$\nabla_{\tilde{w}} J(\boldsymbol{w_O}) = 0 \iff \nabla_w J(\boldsymbol{w_O}) = \nabla_b J(\boldsymbol{w_O}) = 0. \tag{8.11}$$

Quindi le condizioni di Karush–Kuhn–Tucker diventano:

- annullamento della lagrangiana:

$$\begin{cases} \nabla_w J(\boldsymbol{w_O}) = 0 \Rightarrow \boldsymbol{w_O} = \displaystyle\sum_{i=1}^{N} \alpha_i d_i \boldsymbol{x_i} \\ \nabla_b J(\boldsymbol{w_O}) = 0 \Rightarrow \displaystyle\sum_{i=1}^{N} \alpha_i d_i = 0; \end{cases} \tag{8.12}$$

- rispetto dei vincoli: $d_i\left(\boldsymbol{w_O}^T \boldsymbol{x_i} + b_O\right) \geq 1,\ i = 1, \ldots, N$;
- moltiplicatori positivi: $\alpha_i \geq 0$;
- moltiplicatori nulli dei vincoli inattivi: se x_s non è un vettore di supporto, allora $\alpha_s = 0$.

Si ottiene dunque

$$\boldsymbol{w_O} = \sum_{S} \alpha_s d_s \boldsymbol{x_s} \tag{8.13}$$

dove la somma è fatta sull'insieme S dei vettori di supporto.

Ci si riduce dunque a trovare i giusti moltiplicatori α_i. Si può mostrare che esistono tecniche veloci per risolvere questo problema che passano per il problema duale. Ciò che è degno di nota è che, una volta conosciuti i moltiplicatori, si è in grado di classificare nuovi punti $\boldsymbol{x}$ in input anche senza calcolare esplicitamente $\boldsymbol{w_O}$:

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\boldsymbol{w_O}^T \boldsymbol{x} + b_O\right) = \operatorname{sgn}\left(\sum_{S} \alpha_s d_s \boldsymbol{x_s}^T \boldsymbol{x} + b_O\right). \tag{8.14}$$

8.1.3 Il teorema di Vapnik

Avevamo osservato che la VC-dimension nel caso di problemi di classificazione binaria con spazio delle ipotesi formato da iperpiani affini è pari a $m+1$, dove m è la dimensione dello spazio considerato.

Consideriamo ora lo spazio delle ipotesi H_ρ formato dagli iperpiani di separazione con margine di separazione maggiore o uguale a ρ. Siccome stiamo riducendo sensibilmente lo spazio delle ipotesi, è naturale aspettarsi che la VC-dimension sia ridotta.

Questo è confermato dal seguente teorema.

Teorema 8.6. (Vapnik). *Sia D il minimo diametro di una palla che racchiude i punti del training set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^m$. Allora*

$$VC(H_\rho) \leq \min\left\{\left\lceil \frac{D^2}{\rho^2} \right\rceil, m\right\} + 1. \tag{8.15}$$

8.1.4 Soft margin

Osservazione 8.7. L'approccio SVM ai problemi di separazione è particolarmente elegante per via della convessità del problema, dell'esistenza del problema duale, ma soprattutto per via della mancanza di *hyperparameters* nello spazio delle ipotesi: l'algoritmo provvede autonomamente (nel *training process*) a massimizzare il margine di separazione ρ e a minimizzare di conseguenza la VC-dimension.

Tuttavia fa pesante affidamento sull'assunzione che i punti più vicini alla zona limite (cioè i vettori di supporto) siano poco affetti da errori, dato che la soluzione $\boldsymbol{w_0}$ del problema si esprime solo in termini di tali punti.

Per fare a meno di un'assunzione così forte, si può modificare l'SVM e rendere i margini più "morbidi". Più precisamente, si può rendere accettabile che alcuni punti siano all'interno del margine di separazione.

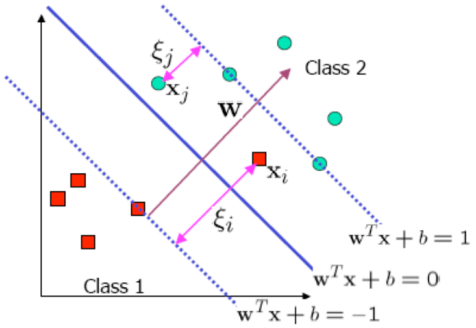

Figura 8.3: Soft Margin e Slack Variables

Definizione 8.8. (Soft Margin SVM). Dati i training samples $\{(\boldsymbol{x_i}, d_i)\}$ con $i = 1, \dots, N$ e $d_i = \pm 1$, e dato un parametro $C \geq 0$, si cerca l'iperpiano di separazione (normalizzato, vedi l'osservazione 8.2) che risolva il problema di minimo (convesso)

$$\min \left\{ \psi(\boldsymbol{w}, \xi_1, \dots, \xi_N) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C \sum_{i=1}^{N} \xi_i \right\} \tag{8.16}$$

sotto i vincoli (lineari)

$$\begin{aligned} d_i(\boldsymbol{w}^T \boldsymbol{x_i}) &\geq 1 - \xi_i \quad \forall i = 1, \dots N \\ \xi_i &\geq 0 \qquad \forall i = 1, \dots N. \end{aligned} \tag{8.17}$$

Osservazione 8.9. Il valore dell'iperparametro C non deve essere troppo grande né troppo piccolo. Nel primo caso vengono tollerati pochi errori, e quindi c'è la possibilità di overfitting. Viceversa, nel secondo caso è probabile l'underfitting.

Osservazione 8.10. L'Hard Margin SVM è il "limite" del problema Soft Margin per $C \to +\infty$.

Definizione 8.11. (Slack Variable). Dati dei punti di training $\{(\boldsymbol{x_i}, d_i)\}$ e un iperpiano con margine di separazione, si dicono *slack variables* le quantità non negative ξ_i che rappresentano di quanto il punto x_i è all'interno del margine (si veda la figura 8.3).

Osservazione 8.12. Attenzione! Non c'è un analogo del teorema di Vapnik con il Soft Margin. In altre parole, se $\tilde{H}_{\rho,\Sigma}$ è lo spazio delle ipotesi formato dagli iperpiani di separazione con margine ρ e che ammettono al più Σ come somma delle slack variables, non c'è nessun buon controllo sulla VC-dimension di $\tilde{H}_{\rho,\Sigma}$.

9 ESEMPI

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

9.1 LEZIONE DI DOMENICA 22 SETTEMBRE

9.1.1 Alcune indicazioni

Un paio di proposte, per uniformità:

- Ogni lezione inizia con `\lecture{⟨data⟩}`, dove la data è del tipo `29/2`.
- Per creare paragrafi numerati, non va usato `\section`, ma `\subsection`;
- Non usate il grassetto, ma il *corsivo*, soprattutto quando s'introduce un *nuovo termine*, e poi si può usare il nuovo termine anche senza corsivo;
- Usate `equation` per le equazioni, in modo che vengano tutte numerate; le equazioni a fine frase terminano con il punto, ma magari evitiamo altro tipo di punteggiatura alla fine di equazioni.
- usare molto gli ambienti `definition`, `theorem`, `example`, `remark`…
- negli elenchi, il ";" alla fine degli *item* e il punto alla fine dell'ultimo. Ma se sono frasi lunghe va bene anche il punto dappertutto.

Definizione 9.1. (Tensore). Un *tensore* è ciò che ruota come un tensore.

$$e^z = \sum_{n=0}^{+\infty} \frac{z^n}{n!}. \tag{9.1}$$

```c
#include <stdio.h>
int main()
{
  // printf() displays the string inside quotation
  printf("Hello, World!");
  return 0;
}
```