6/5 slides 16

Molti modelli possono essere estesi permettendo di apprendere da dataset con non linearità, con l'uso di funzioni non lineari o feature transformation: model(x, Θ) = w0 + f1(x)w1 + … + fB(x)wB

Con f1,..,fB funzioni/feature/caratteristiche non lineari e parametrizzate o non parametrizzate dei dati e w0,..,wB insieme dei pesi Θ.

Il nostro scopo è apprendere tali caratteristiche (quali f1,…,fb usare, la quantità B, i parametri) in modo automatizzato a partire dai dati, invece che crearle a mano.

Supponiamo di conoscere completamente il fenomeno che genera i dati, che sono anche privi di rumore. In tale scenario ideale le caratteristiche esatte sono apprese combinando elementi da un insieme di trasformazioni base detto Universal Approximator. Gli UA sono un insieme di f di una certa famiglia che approssimano altre funzioni, a prescindere dalla loro complessità. Sono un gruppo di diversi tipi di f che possono, dopo essere stati opportunamente modulati tramite la stima dei loro parametri, approssimare modelli di qualunque complessità.

Supponiamo di avere un insieme B di vettori f di lunghezza N. B è detto spanning set, e lo span di B è l'insieme di tutte le possibili combinazioni lineari dei vettori che lo compongono: span B = span{f} = f1w1 + f2w2 + … + fBwB = y

E se le f sono non lineari: span B = span{f(x)} = w0 + f1(x)w1 + f2(x)w2 + … + fB(x)wB = y(x) con w0=bias. L'obiettivo è trovare i pesi tali che f1w1 + f2w2 + … + fBwB ~ y, in cui l'approssimazione è valutata con:

- Diversità
- Valore di B } Rango
- Capacità di approssimazione per mezzo della minimizzazione della loss

Le funzioni di spanning possono essere parametrizzate, cioè definite per mezzo di parametri che comunque possiamo stimare.

Se prendiamo B>=N vettori per definire lo spanning set, con almeno N vettori linearmente indipendenti, allora tale spanning set è uno spazio vettoriale con base costituita dagli N vettori indipendenti detto Universal Approximator e tramite esso possiamo approssimare qualsiasi vettore N-dimensionale scegliendo un'opportuna combinazione di elementi di B.

Famiglie di approssimatori:

- Forma fissa: funzioni non lineari prive di parametri (es polinomi: f1(x)=x, f2(x)=x^2, f3(x)=x^3,…). Un polinomio di grado D è la combinazione dei primi D polinomi fino a quello di grado D.
- Basate su reti neurali: ne fanno parte le funzioni di attivazione all'interno delle quali c'è una qualche espressione di x. Sono quindi funzioni parametriche.
- Basate su alberi: famiglia delle funzioni a gradino discrete.

Un Approximator lavora con una sola famiglia di approssimatori, così che ogni famiglia abbia i suoi procedimenti di addestramento/ottimizzazione da usare.

10/5

Validazione:

Naive Cross-Validation: si usa la validation per valutare modelli via via di complessità maggiore, generati aggiungendo sequenzialmente una unità della famiglia di approssimatori, e quindi un insieme di M modelli:

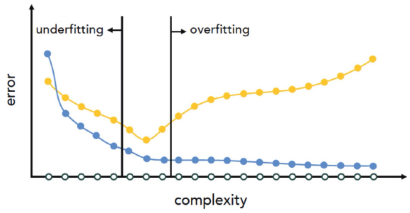
model1(x, Θ1) = w0 + f1(x)w1

model2(x, Θ2) = w0 + f1(x)w1 + f2(x)w2

…

modelM(x, ΘM) = w0 + f1(x)w1 + … + fM(x)wM

Cioè {model m(x, Θm)} con m che va da 1 a M. Valutando questi modelli al crescere della complessità si ottengono questi andamenti di training e di validation:



Cross-Validation via Boosting: si considera un modello a cui si aggiunge un'unità per volta, oppure si considera un modello completo model(x, Θ) = w0 + f1(x)w1 + … + fM(x)wM di cui si ottimizzano i parametri di ogni unità, una unità alla volta.

Round 0: model0(x, Θ0) = w0 ; si ottimizza $\frac{1}{P}\sum_{p=1}^{P}(model_0(x_p,\Theta_0) - y_p)^2 = \frac{1}{P}\sum_{p=1}^{P}(w_0 - y_p)^2$

Round 1: model1(x, Θ1) = model0(x, Θ0) + fs1(x)w1 ; si ottimizza $\frac{1}{P}\sum_{p=1}^{P}(model_0(x_p,\Theta_0) + f_{s1}(x_p)\,w_1 - y_p)^2 = \frac{1}{P}\sum_{p=1}^{P}(w_0^* + f_{s1}(x_p)\,w_1 - y_p)^2$ ↗ Giá ottimizzato

Round m>1: model m(x, Θm) = model m-1(x, Θm-1) + fsm(x)wm ; si ottimizza $\frac{1}{P}\sum_{p=1}^{P}(model_{m-1}(x_p,\Theta_{m-1}) + f_{sm}(x_p)\,w_m - y_p)^2$

$\frac{1}{P}\sum_{p=1}^{P}(w_0^* + w_1^* f_{s1}(x_p) + \dots + w_{m-1}^* f_{s\,m-1}(x_p) + f_{sm}(x_p)\,w_m - y_p)^2$

Residuals: un residuo è ciò che rimane da rappresentare dopo aver sottratto dal valore stimato ciò che è già stato appreso dal m-1 esimo modello. Partiamo dalla funzione di costo g e dal modello boosted model m:

$g(\theta_m) = \frac{1}{P}\sum_{p=1}^{P}(model_m(x_p,\theta_m) - y_p)^2$ e $model_m(x_p,\theta_m) = model_{m-1}(x_p,\theta_{m-1}) + f_m(x_p)\,w_m$

Riscrivo: $g(\theta_m) = \frac{1}{P}\sum_{p=1}^{P}(f_m(x_p)\,w_m - (y_p - model_{m-1}(x_p)))^2$

Che vogliamo minimizzare stimando i parametri della singola unità aggiuntiva in modo che valga:

$f_m(x_p)\,w_m \approx y_p - model_{m-1}(x_p) = residuo$

Early stopping: Se partiamo da un modello e lo addestriamo, ad esempio via Boosting, sappiamo che oltre un certo numero di unità aggiunte tale modello andrà in overfitting. Vogliamo quindi fermarci prima, ma quando? In teoria, dato l'andamento del grafico sopra, dobbiamo fermarci appena il validation error inizia a salire. Nella pratica però abbiamo bisogno di avere la sicurezza di fermarci in una condizione in cui la predizione sia buona: si tiene conto dell'ultima volta in cui il modello ha migliorato le sue prestazioni (cioè il validation error è diminuito) e ci si ferma dopo un certo numero di epoche aggiuntive in cui non ci sono stati miglioramenti significativi (patience criteria).

Regolarizzazione: si tratta di funzioni che si aggiungono alla loss, pesate da un parametro lambda: $costo = g(\theta) + \lambda h(\theta)$

Questa funzione è detta costo regolarizzato, e serve a spostarci, con un certo criterio, da una famiglia di parametri ottimali per adattarci meglio al nostro problema.

Bagging Cross-Validation Models: si creano diversi split di training e validation, determinando un modello appropriato su ciascuna suddivisione e quindi si fa una "media" dei modelli ottenuti. Con media si intende mediana, moda o norma tra gli output dei diversi modelli per essere più sensibili rispetto alla varianza nei dati. Solo in questo caso, siccome i modelli sono addestrati separatamente, possono appartenere a famiglie di approssimatori diverse.

13/5 slides 17

Metodi Kernel: si basano sull'idea di trovare una funzione kernel che determini la somiglianza tra punti mappati in uno spazio delle caratteristiche trasformato ad alta densità, dove è più facile identificare i pattern rilevanti (quindi si alza la dimensionalità per identificare meglio i pattern). Le funzioni kernel comunemente utilizzate sono:

- Lineari
- Polinomiali
- A base radiale (RBF)
- Sigmoidi

Kernel Trick: consente dagli algoritmi di calcolare prodotti scalari tra punti nello spazio delle caratteristiche ad alta dimensionalità senza trasformare esplicitamente i dati. Il kernel trick "bypassa" le caratteristiche aggiuntive che abbiamo scelto per evidenziare i pattern.

SVM: metodi kernel tra i più utilizzati

Gaussian Process: forniscono un framework probabilistico

Approssimatori Fixed-Shaped: non ci sono parametri interni, sono distinti in base al grado o a qualche indice naturale (es fm(x) = sin (mx))

Basi di Fourier: sono altri approssimatori, sono insieme di onde seno e coseno accoppiate (con frequenza crescente) della forma: f2m-1(x)=sin(2πmx), f2m(x)=cos(2πmx) con m >=1, o in forma compatta complesso-esponenziale

Per un input generico di dimensione N ogni unità di Fourier multidimensionale ha forma $f_m(x) = e^{2\pi i m x}$ $f_m(x) = e^{2\pi i m_1 x_1} \dots e^{2\pi i m_N x_N}$

Con approssimatori fixed form in una generica unità il numero di fattori cresce esponenzialmente con la dimensione dell' input N poiché per costruire un modello polinomiale di grado D vengono inclusi nella stessa configurazione tutti i termini con 0 <= m1+m2+…+mN <= D, con m1,m2,…,mN >= 0.

Kernelization: mira a costruire feature a forma fissa per qualsiasi problema di apprendimento automatizzato, evitando il problema dell'esplosione combinatoria, con nuove features a formula fissa definite esclusivamente attraverso una rappresentazione kernelizzata.

Abbiamo quindi approssimatori, modello e funzione di costo, e vogliamo riproporre questi elementi sotto il kernel.

Prendiamo un approccio supervisionato, con funzioni di costo Least Squares e Softmax. Il generico modello è:

model(xp,w) = w0 + f1(xp)w1 + … + fM(xp)wM che posso riscrivere con w: vettore di feature touching weights:

model(xp, b, w) = b + f_p^Tw , con b = w0, w=[w1, …, wM]$_T$

Partendo da fp = [f1(xp), … , fM(xp)]$_T$scrivo il caso Least Squares come:

$g(b,w) = \frac{1}{P}\sum_{p=1}^{P}(b + f_p^T w - y_p)^2 = \frac{1}{P}\sum_{p=1}^{P}(b + f_p^T(Fz + r) - y_p)^2 = \frac{1}{P}\sum_{p=1}^{P}(b + f_p^T Fz - y_p)^2$ con $w = Fz + r$ decomposizione di w

Introducendo la matrice simmetrica $H_{P\times P} = F^T F$ detta Kernel Matrix dove la p-esima colonna è $h_p = F^T f_p$

definiamo il costo kernelizzato: $g(b,z) = \frac{1}{P}\sum_{p=1}^{P}(b + h_p^T z - y_p)^2$

e il corrispondente modello valutato sulla p-esima istanza in input come model(xp, b, z) = b + hp$_T$z

La matrice F è una matrice di approssimatori (che "tirano fuori" le features). Il fulcro di questa tecnica è che tramite questi passaggi "scompaiono" le f e rimangono le h, tramite il kernel che fa il mapping dei punti del dominio.

16/5

La kernel function accetta due istanze nello spazio dimensionale originale e restituisce il prodotto scalare delle istanze nello spazio kernel delle features di dimensionalità superiore:

$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$ con $\Phi: \mathbb{R}^N \to \mathbb{R}^M$ funzione di mapping tra i 2 spazi

Una kernel matrix è simmetrica e positiva semidefinita, cioè $x^T H x >= 0$ per ogni $x \in \mathbb{R}^p$ e definisce il punto esatto di ogni coppia di istanze nello spazio a dimensionalità superiore.

Kernel Polinomiale:

- Si consideri la seguente mappatura polinomiale di grado D=2 dallo spazio in input N=2, a quello M=5 dimensionale data dal seguente vettore di trasformazione delle caratteristiche: f=[x1, x2, $x1_2$ x1x2, $x2_2$]$_T$
- Se si moltiplicano alcune o tutte le componenti di f per un valore costante $\sqrt{2}$ il processo di trasformazione delle feature non viene alterato poiché tali valori vengono assorbiti dai pesi associati in w quando si forma il modello (x,b,w)=b+f w
- Se si denotano con u=xi e v=xj la i-ma e j-ma istanza in $\mathbb{R}^2$ in input, l'elemento (i, j) della matrice kernel per un polinomio di grado D=2 sarà il seguente: $H = F^T F$

$$h_{i,j} = f_i^T f_j = \begin{bmatrix} \sqrt{2}u_1 & \sqrt{2}u_2 & u_1^2 & \sqrt{2}u_1u_2 & u_2^2 \end{bmatrix} \begin{bmatrix} \sqrt{2}v_1 \\ \sqrt{2}v_2 \\ v_1^2 \\ \sqrt{2}v_1v_2 \\ v_2^2 \end{bmatrix}$$ = svolgo e aggiungo +1-1 = $(1 + u_1v_1 + u_2v_2)^2 - 1 = (1 + u^Tv)^2 - 1$

- Perciò la matrice kernel H può essere costruita senza passare per le caratteristiche esplicite ma definendola entry wise come:
- Generalizzando per N e D generici, con N = dimensione delle rappresentazioni in input e D = dimensione dello spazio superiore: $h_{i,j} = (1 + x_i^T x_j)^2 - 1$

Kernel di Fourier:

$h_{i,j} = (1 + x_i^T x_j)^D - 1$

Una trasformazione D-Fourier delle caratteristiche con N=1 e M=2D può essere scritta come vettore di caratteristica 2Dx1

$$f = \begin{bmatrix} \sqrt{2}\cos(2\pi x) \\ \sqrt{2}\sin(2\pi x) \\ \sqrt{2}\cos(2\pi Dx) \\ \sqrt{2}\sin(2\pi Dx) \end{bmatrix}$$

in cui la radice di 2 non altera la trasformazione originale. Il corrispondente elemento i,j della kernel matrix può essere scritto come:

$h_{i,j} = f_i^T f_j = \sum_{m=1}^{D} \left[2(\cos(2\pi m x_i)\cos(2\pi m x_j) + \sin(2\pi m x_i)\sin(2\pi m x_j)) \right]$

Impiegando l'identità cos(a)cos(b) + sin(a)sin(ab) = cos(a-b) si ha:

Tramite la rappresentazione complessa si ha:

$h_{i,j} = \sum_{m=1}^{D} 2\cos(2\pi m(x_i - x_j))$

nel caso in cui xi-xj $\in$ Z, altrimenti si fattorizza a:

$h_{i,j} = \sum_{m=-D}^{D} \left[e^{2\pi i m(x_i - x_j)} + e^{-2\pi i m(x_i - x_j)} \right] = \left[\sum_{m=-D}^{D} e^{2\pi i m(x_i - x_j)} \right] - 1$

Notando che la somma sul lato destro è una serie geometrica si ha:

$h_{i,j} = \left[e^{-2\pi i D(x_i - x_j)} \sum_{m=0}^{2D} e^{2\pi i m(x_i - x_j)} \right] - 1$

che è il generico coefficiente della kernel matrix, generalizzabile a input N dimensionali.

$h_{i,j} = \frac{\sin((2D+1)\pi(x_i - x_j))}{\sin(\pi(x_i - x_j))} - 1$

Kernel Radial Basis Function (RBF):

$h_{i,j} = e^{-\beta \|x_i - x_j\|_2^2} = e^{\frac{\|x_i - x_j\|_2^2}{2\delta^2}}$

Con beta parametro da definire in base al dataset oppure espressa tramite varianza. Questi due iper parametri danno la misura di similarità tra istanze. È una kernel matrix sfruttata per feature che hanno una distribuzione Gaussiana. I kernel RBF sono la forma più generalizzata di kernelizzazione e sono tra i kernel più utilizzati avendo caratteristiche che si avvicinano a quelle di una distribuzione gaussiana.

23/5 slides 18

Dimensionality Reduction: è l'estrazione delle features tramite la trasformazione/proiezione del contenuto informativo dei dataset in un nuovo sottospazio di features con dimensionalità inferiore all'originale.

Principal Component Analysis (PCA): è una tecnica di trasformazione lineare non supervisionata. Le Componenti Principali sono i vettori, ortogonali tra loro, che costituiscono il nuovo spazio che viene identificato tramite le direzioni di massima varianza nello spazio dei dati originale.

La proiezione del nuovo spazio è realizzata tramite una matrice di trasformazione/proiezione $W \in \mathbb{R}^{d \times k}$ che consente di mappare un vettore x=(x1, … , xd) $\in \mathbb{R}^{d}$ in un nuovo spazio di caratteristiche k-dimensionale, con k<d, con generico vettore z=(z1, … , zk) $\in \mathbb{R}^{k}$ tale che xW = z.

La prima componente principale è quella a massima varianza, poi a seguire quelle con varianza discendente con la condizione che siano ortogonali alle componenti principali precedenti.

È obbliga standardizzare le features.

Varianza spiegata: È la varianza spiegata dalla retta di regressione ed è la media della distanze al quadrato tra i valori e la retta costante.

(Richiami su autovalori, autovettori, polinomio caratteristico, equazione caratteristica)

La PCA sfrutta la matrice di covarianza [cov(x,y)] o la matrice di correlazione (es di Pearson $\varphi_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y}$).

Se decomponiamo una matrice di covarianza nei suoi autovalori e autovettori, l'autovettore associato all'auto valore più alto corrisponde alla direzione di massima varianza nel set di dati. In generale gli autovettori corrispondono alle PC, gli autovalori alle varianze.

La procedura PCA è costituita dai seguenti passi:

1. *Standardizzare* il dataset ***d***-dimensionale (tipicamente in modo che la varianza sia 1)
2. Costruire la *matrice di covarianza*.
3. Decomporre la matrice di covarianza in *autovalori* e *autovettori*.
4. Ordinare il valore degli *autovalori* e selezionare i ***k*** autovettori corrispondenti ai valori più alti.
5. Costruire la *matrice di proiezione* $\mathbf{W} \in \mathbb{R}^{d \times k}$ con colonne corrispondenti ai ***k*** autovettori.
6. Proiettare il dataset in input nel nuovo sottospazio: $\mathbf{xW} = \mathbf{z}$

27/5 slides 19

Altre due tecniche di riduzione della dimensionalità sono l'LDA e il t-SNE.

Linear Discriminant Analysis (LDA): La differenza da PCA è che LDA è un algoritmo supervisionato, è una trasformazione lineare che mira a trovare il sottospazio delle caratteristiche che ottimizza la separabilità delle classi. È basato su delle hp:

- Che i dati siano distribuiti normalmente
- Che le classi abbiano matrici di covarianza identiche
- Che gli esempi di training siano statisticamente indipendenti tra loro

Tuttavia funziona bene anche se questi vincoli vengono rilassati.

- Standardizzare il dataset iniziale ***d***-dimensionale
- Per ogni classe, determinare il vettore ***d***-dimensione costituito dal valor medio ricavato per ogni dimensione.
- Costruire la **matrice di dispersione tra classi** (**between-class scatter matrix**) S_B, e la **matrice di dispersione all'interno delle classi** (**in-between** o **within-class scatter matrix**) S_W.
  - Una **scatter matrix** è una stima della *matrice delle covarianze* costruite a partire dai dati estratti da un dataset.
- Calcolare gli *autovettori* e i corrispondenti *autovalori* della matrice $S_W^{-1}S_B$
- *Ordinare* gli autovalori in modo decrescente ,e di conseguenza, i relativi autovettori.
- Scegliere i **k** autovettore che corrispondono ai **k** autovalori di valore più elevato, e costruire una *matrice di proiezione (o trasformazione)* $\mathbf{W} \in \mathbb{R}^{d \times k}$, con colonne pari agli autovettori.
- Proiettare il dataset in input nel nuovo sottospazio: $\mathbf{xW} = \mathbf{z}$

$\longrightarrow$ $S_i = \sum_{x \in D_i} (x - m_i)(x - m_i)^T$ $\forall$ classe i

$S_W = \sum_{i=1}^{c} S_i$

$S_B = \sum_{i=1}^{c} (m_i - m)(m_i - m)^T$ con m = vettore media

Manifold Learning: è un tipo di tecnica non lineare per la riduzione della dimensionalità, in cui le istanze originali sono proiettate in spazi di dimensioni ridotte chiamate Latent Manifolds. Le proiezioni devono catturare la struttura dei dati originali per poterli preservare nello spazio latente.

T-Distributed Stochastic Neighbour Embedding: è una tecnica non lineare di riduzione della dimensionalità per insiemi di dati multidimensionali in spazi a 2 o 3 dimensioni.

Caratterizza le istanze originali in base alla distanza misurata a coppie mediante un kernel Gaussiano.

Definisce una distribuzione di probabilità di tale distanza nel nuovo spazio latente che più si avvicina alla distribuzione nello spazio originale, mirando quindi a preservare tali distanze minimizzando la divergenza di 2 distribuzioni con un approccio gradient descent.

Non crea una funzione o matrice di proiezione ma opera unicamente sui dati originali, per cui non può essere utilizzata su nuove istanze (perché non creando ad esempio una matrice, non è possibile applicarla tramite xW=z)

(t-SNE step by step)

31/5 slides 2

Time Series: è una rappresentazione per dati sequenziali, ovvero dati in cui l'ordine degli elementi in ingresso (istanze) è rilevante. Tipicamente i campioni di una sequenza dipendono l'uno dall'altro.

I modelli che li elaborano sono progettati per gestire dati non distribuiti in modo indipendente e identico (non IID), e seguono generalmente tre approcci:

- Apprendimento Sequenziale Supervisionato: ad esempio il Part Of Speech Tagging in cui i dati sono organizzati come $\{(xi, yi)\}_{i=1}^{N}$ in cui ogni istanza è una coppia di sequenze (xi, yi) con xi=<xi1, … , xiTi>, yi=<yi1, … , yiTi>, per esempio xi=< do you want fries with that> e yi=< verb pronoun verb noun prep pronoun >, e l'obiettivo è arrivare ad un classificatore supervisionato h tale che y=h(x) con x nuova sequenza. Con questo approccio quindi vogliamo predire tutti i valori di y.
- Previsione: questo approccio è correlato al precedente; l'obiettivo è prevedere il t+1esimo elemento in una sequenza <y1, … , yt>. Possono anche essere disponibili altre caratteristiche <x1, … , xt, xt+1> ma comunque sempre e solo fino al tempo corrente t+1 per cui si sta cercando di fare una previsione. Con questo approccio quindi abbiamo i valori reali per i primi t valori di y osservati e vogliamo predire il t+1esimo.
- Classificazione della sequenza: ho una sequenza e la voglio classificare, cioè applicare una singola etichetta y all'intera sequenza di input <x1, … , xt>. In questo tipo di problema quindi ogni esempio di addestramento consiste in una coppia (xi, yi) in cui xi è una sequenza di input <xi1, … , xiTi> e ogni yi è una label di classe.

Feature Selection e Long Distance Interaction: la long distance interaction si occupa di identificare dei pattern che tengono traccia delle correlazioni su lungo periodo di tempo, ovvero di eventi/istanze che si influenzano dopo molto tempo. La feature selection in ambiti di questo genere può seguire tre approcci:

- Approccio wrapper: si ha un dataset completo su un intervallo di tempo ampio. Si generano vari sottoinsiemi di caratteristiche e si valutano eseguendo su di essi l'algoritmo di apprendimento e misurando l'accuratezza del classificatore risultante per ogni sottoinsieme.
- Si considerano tutte le feature e poi si impiega una misura di penalty per ridurre il peso delle feature poco rilevanti
- Si considera una misura di feature relevance
- Sliding Windows: si considera una finestra di feature (la dimensione di questa sezione estratta dalla sequenza time series è fissa) e si addestra il classificatore su questa finestra < xi t-d, …, xi t, … , xi t+d > . Per le sequenze iniziali e finali i valori mancanti sono impostati a zero (padding). Il classificatore a questo punto è addestrato convertendo ogni istanza di addestramento sequenziale (xi, yi) in istanze multiple e usando un apprendimento supervisionato standard. La y risultante è correlata solo con la finestra temporale considerata quindi come previsione può non essere sufficientemente long-distance, in quanto può non tenere conto di correlazioni abbastanza lontane nel tempo. Il vantaggio nell'uso di questo metodo sta però nella possibilità di gestire serie molto lunghe di dati e di riutilizzare modelli già visti su tali serie "ridotte".
- Recurrent sliding windows: il valore previsto yit è inserito come input per supportare la previsione di yi t+1. Cioè le previsioni: < yi t-d, … , yi t-1 >
  sono usate come input insieme alla sliding window: < xi t-d, … , xi t-1, xi t , xi t+1, … , xi t+d >
  Questo processo può essere iterato in modo che le previsioni di ogni iterazione siano utilizzate come input dell'iterazione successiva.

Features di Time Series:

- Data e ora
- Log features
- Window features
- Time until next event / time since last event

3/6 slides 21

Hyperparameter tuning (HPs-T): nell'addestramento ML ogni categoria di modello e particolare dataset impiegato necessitano di una diversa configurazione di iperparametri. Nell'HPs-T si sperimentano diversi set di parametri, in modo manuale o automatico, e si analizzano le prestazioni selezionando la configurazione più adatta. Nello split training-test-validation il validation set è quello tipicamente usato per questo tuning. Ricordiamo che gli iperparametri sono model independent e controllano il processo di training nel senso che ne determinano alcune caratteristiche di velocità e complessità e sono fondamentali per evitare overfitting e underfitting. Le categorie di iperparametri sono:

- Model HPs: sono coinvolti nella struttura del modello e definiscono il costrutto di un modello stesso. Ad esempio, gli iperparametri del modello nel deep learning possono essere attributi di una rete neurale, come il numero di unità nascoste, gli strati, la dimensione del filtro, il padding…
- Optimiser HPs: utilizzati nel processo di ottimizzazione. Si chiamano iperparametri ottimizzatori perché ottimizzano le prestazioni del modello dettando il modo in cui il modello apprende i modelli nei dati. Esempi di iperparametri ottimizzatori nel deep learning sono l'algoritmo di ottimizzazione, il tasso di apprendimento…
- Data HPs: riguardano i dati da utilizzare per l'addestramento. Spesso i dati non sono sufficienti, non sono soddisfacenti in termini di variazione o distribuzione. Si impiegano tecniche quali data splitting, data sampling, e data augmentation (es. resizing, cropping, binarization, etc). Rientra anche il valore di k nella k-fold cross-validation.

Degli esempi di iperparametri in una rete neurale possono essere il learning rate, il learning rate decay, il momentum, il numero di nodi o layer nascosti, la dimensione del mini batch di training, le epoche…

La HPs Optimization è formalizzata come segue: $x^* = \text{argmin } f(x)$ con x appartenente ad X

La f(x) rappresenta la funzione obiettivo valutata sul validation set. x^* è l'insieme di iperparametri che minimizza f e può assumere qualsiasi valore nello spazio degli iperparametri X. La valutazione della funzione obiettivo è computazionalmente onerosa e richiede molto tempo, per cui il processo di tuning non può essere svolto manualmente ma va automatizzato, seguendo uno tra gli approcci principali

- Grid search: specifichiamo un elenco di iperparametri e una metrica di valutazione. L'approccio itera su tutte le combinazioni possibili per determinarne la migliore. La ricerca a griglia funziona bene, ma essendo esaustiva rispetto all'elenco definito, è computazionalmente intensa, soprattutto con un gran numero di iperparametri da considerare. Definiamo una griglia di valori, che corrisponde allo spazio degli iperparametri X. L'algoritmo esplora esaustivamente lo spazio in modo sequenziale e addestra un modello per ogni possibile combinazione di valori di iperparametri. Il numero di modelli da addestrare cresce esponenzialmente all'aumentare del numero di iperparametri da considerare nello spazio, il che rende la ricerca poco efficiente, sia in termini di potenza di calcolo che di tempo.
- Random search: simile alla grid search, seleziona gruppi di iperparametri in modo casuale a ogni iterazione. Funziona bene quando un numero relativamente piccolo di iperparametri determinano il risultato del modello. Non viene fornito un insieme esplicito di valori possibili per ogni iperparametro, bensì una distribuzione statistica per ogni iperparametro da cui sono campionati i valori in modo stocastico. È più facile controllare o limitare il numero di combinazioni di iperparametri da generare. Ad esempio, possiamo specificare di addestrare solo un numero fisso di modelli. Il numero di iterazioni di ricerca può essere limitato anche in base al tempo o alle risorse disponibili.

• Bayesian Optimization: basata sul teorema di Bayes, che descrive la probabilità che un evento si verifichi in relazione alle conoscenze attuali/acquisite, e quindi sul principio alla base dell'inferenza Bayesiana: più informazioni ci sono, più la stima è corretta. Quando è applicata all'ottimizzazione degli iperparametri, l'algoritmo costruisce un modello probabilistico da un insieme di iperparametri che mira ad ottimizzare una metrica predefinita. Si tiene traccia dei risultati delle valutazioni passate per stimare un modello probabilistico chiamato surrogato che mette in relazione i valori degli iperparametri con le performance ottenute dalla funzione obiettivo: P(score | hyperparameters), più facile e veloce da ottimizzare rispetto alla funzione obiettivo originale. Quindi la bayesian optimization mira a trovare il successivo set di iperparametri da valutare rispetto alla funzione obiettivo originale, cercandoli attraverso la funzione surrogata, evitando quanto possibile di valutare l'intero modello da ottimizzare.

Procedimento

- *Costruire un modello di probabilità surrogato* della funzione obiettivo
- *Stimare gli iperparametri* in base al modello surrogato
- *Valutare* la configurazione con tali iperparametri con la funzione obiettivo
- *Aggiornare il modello surrogato* incorporando i nuovi risultati
- Ripetere i passaggi fino al *criterio di stop*

• Sequential Model-Based Optimization: è una formalizzazione della bayesian optimization, dove sequenziale si riferisce all'esecuzione iterativa, provando ogni volta iperparametri migliori ricavati seguendo l'approccio bayesiano e aggiornando il modello surrogato di conseguenza. L'approccio è basato sui seguenti aspetti:

1. Uno spazio di iperparametri su cui effettuare la ricerca
2. Una funzione obiettivo del modello originale che prende in considerazione gli iperparametri e restituisce uno score da minimizzare
3. Una funzione surrogata della funzione obiettivo, cioè la rappresentazione della probabilità della funzione obiettivo in base alla configurazione degli iperparametri (esistono diverse varianti di questo metodo che si differenziano per la costruzione della funzione surrogato)
4. Una cronologia di {< score, iperparametro >} che l'algoritmo impiega per aggiornare il modello surrogato

Selection Function: è il criterio in base al quale viene scelto il successivo insieme di iperparametri della funzione surrogata. Nel caso più noto, la funzione corrisponde alla Expected Improvement: $EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)\,dy$
Dove y^* è un valore soglia della funzione obiettivo, x è l'insieme candidato di iperparametri, y è il valore effettivo della funzione obiettivo utilizzando gli iperparametri x, mentre p(y|x) è il modello surrogato. L'obiettivo è massimizzare l'utilità attesa, ovvero EI, scegliendo il candidato migliore tra le x.
Tree-structured Parzen Estimator (TPE): Invece di impiegare un modello basato su un processo gaussiano, non rappresenta direttamente p(y|x) ma costruisce il modello con la regola di Bayes: $p(y|x) = \frac{p(x|y)\cdot p(y)}{p(x)}$
dove p (x|y) è la probabilità degli iperparametri x dato il punteggio della funzione obiettivo, che è a sua volta espressa come: p(x|y):

$$\begin{cases} l(x) & \text{se } y < y^* \\ g(x) & \text{se } y \geq y^* \end{cases}$$

dove $y<y^*$ rappresenta un valore della funzione obiettivo inferiore alla soglia. La spiegazione di questa equazione è che si creano 2 diverse distribuzioni: una indica la distribuzione degli iperparametri l(x) quando lo score è inferiore alla soglia, e un'altra in cui il valore della funzione obiettivo è maggiore della soglia g(x). La funzione surrogata EI che otteniamo dipende dal rapporto l(x)/g(x) quindi per massimizzare EI dobbiamo generare iperparametri che sono più probabili sotto la distribuzione l(x) rispetto a g(x). Perciò i punti da esplorare saranno concentrati nella zona caratterizzata da valori sotto soglia. Sceglieremo iperparametri corrispondenti a valori più alti del rapporto l(x)/g(x) e tali iperparametri verranno valutati con la funzione obiettivo. Se la funzione surrogata è corretta, tale configurazione genererà un miglioramento. Il criterio descritto permette di bilanciare exploration ed exploitation.