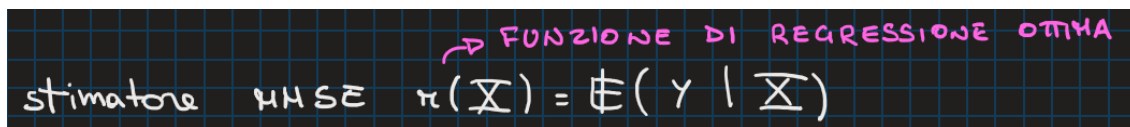


Regressione non parametrica - 11/11 + 12/11

Recap

Il **primo problema** che abbiamo inquadrato è una regressione nella quale il modello statistico è perfettamente conosciuto, con X che è input o feature e Y è output o label.

In un contesto di regressione nel quale il modello è perfettamente noto lo stimatore migliore è MMSE, cioè la media a posteriori, la media di Y dato X , da non confondersi con MSE che è Mean Squared Error, che è una misura dell'errore.



stimatore MMSE $\pi(X) = E(Y | X)$ → FUNZIONE DI REGRESSIONE OTTIMA

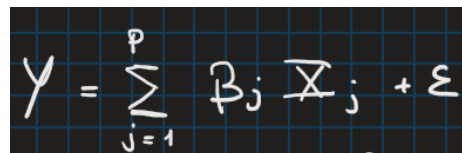
Per calcolare la media statistica di $Y|X$ ci sono varie possibilità:

- mi è stata fornita;
- conosco il modello (quello di $f_{Y|X}$).

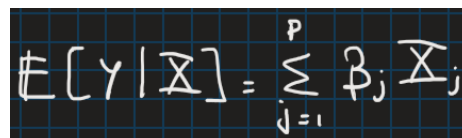
Il **secondo problema** che abbiamo inquadrato è una regressione nella quale il modello statistico non è perfettamente conosciuto e quindi si cerca di impararlo da un data-set.

Noi nello specifico abbiamo visto la **regressione lineare**.

Regressione lineare significa che abbiamo postulato il seguente modello:


$$Y = \sum_{j=1}^p \beta_j X_j + \epsilon$$

Per questo modello lo stimatore MMSE sarà il seguente (lo abbiamo dimostrato qui):


$$E[Y | X] = \sum_{j=1}^p \beta_j X_j$$

Assumendo


$$E[\epsilon | X] = 0$$

Quindi se conoscessimo perfettamente β conosceremmo anche lo stimatore ottimo, il nostro modello da apprendere è perfettamente conosciuto.

Invece quando studiamo la regressione lineare con un data-set noi non conosciamo β e quindi tutto il lavoro che facciamo corrisponde ad imparare i parametri β che sono i parametri del modello.



Il nostro obiettivo è ottenere la $r(x)$, per ottenerla servono le β ma è sbagliato dire che il nostro obiettivo è ottenere le β .

Se il modello mi fosse stato rivelato io non avrei avuto bisogno di stimare i parametri del modello.

Distinzione in fasi

Distinguiamo tra la fase di addestramento e un'altra fase che possiamo chiamare predizione, chiamarla testing o test imporrebbe di pensare alla differenza tra validation e test, invece la chiamiamo predizione perché quello che andiamo a fare, non su un data-set (ricordiamo che viene diviso in training set, validation set e test set) ma nel mondo reale su dati e osservazioni del futuro, in pratica la fase di predizione è quando il modello è pronto ed è veramente usato sul campo.

Quando siamo in fase di predizione aggiungiamo il pedice 0 alle variabili.

Supervised setting X

Se il modello non è noto servono dati e si parla di contesto Supervisionato, noi abbiamo infatti studiato la regressione lineare supervisionata.

Si parla di supervisionata perché abbiamo un training set con tante coppie ingressi-uscita, da chi è supervisionata la regressione? Dal training set e noi impariamo dal training set.

Si parla di unsupervised se abbiamo solo le X ma non è solo questo il ragionamento.

La differenza tra MMSE quando tutto è noto e la regressione ideale che abbiamo studiato è che manca il modello ma abbiamo un Training set, la condizione che l'errore è condizionatamente a media nulla e sappiamo che la relazione ingresso-uscita è lineare, il che vuol dire che abbiamo stabilito la forma parametrica della funzione di regressione, quindi non abbiamo semplicemente un training set, abbiamo un training set per imparare una funzione che ha una forma lineare.

In merito all'errore:

$$\begin{aligned} \varepsilon &= Y - \pi(x) \Rightarrow Y = \pi(X) + \varepsilon \\ E[\varepsilon | X] &= 0 \end{aligned}$$

Si può dire che la label è uguale alla funzione di regressione più l'errore che ha la caratteristica specificata.

Questa forma è generale mentre la seguente è particolare.

$$Y = \sum_{j=1}^p \beta_j X_j + \varepsilon$$

La faccia è la stessa salvo che la sommatoria βX è una particolare $r(x)$ alla quale ho dato una specifica forma.

Il nostro obiettivo è arrivare a rispondere alla domanda, cosa succede se non ci viene detto che è possibile usare questa forma parametrica (lineare) o qualunque altra forma parametrica.

Se nessuno mi da questa funzione parametrica cosa facciamo?

Regressione non parametrica

Vogliamo stimare una Y a partire da alcuni dati X , il modo ottimo per farlo secondo un criterio di errore quadratico medio è lo stimatore media posteriori: la media di Y dato X .

↗ FUNZIONE DI REGRESSIONE OTTIMA

stimatore MMSE $\pi(X) = E(Y | X)$

La media di Y è funzione di Y ? NO.

La media di $Y|X$ è funzione di Y ? NO.

La media di $Y|X$ è funzione di X ? SI.

La media condizionata è funzione del condizionante.

▼ Cose da rivedere

- Definizione di media
- Definizione di media condizionata
- Definizione di media congiunta
- Definizione di covarianza
- Definizione di correlazione
- Definizione di indipendenza
- Definizione di incorrelazione
- Altro? Sugerite nei commenti!!!

$$\begin{aligned}\pi(\bar{X}_0) &= \mathbb{E}[Y_0 | \bar{X}_0] \\ &\quad \hookrightarrow \text{prediction} \\ \varepsilon &= Y_0 - \pi(\bar{X}_0) \quad \Rightarrow \quad Y_0 = \pi(\bar{X}_0) + \varepsilon \\ \mathbb{E}[\varepsilon | \bar{X}_0] &= 0\end{aligned}$$

Il pedice 0 indica che ora siamo in fase di predizione, i pedici da 1 ad N sono del training data-set, ora siamo in predizione. La r se ottenuta dal training set è sporca e non è più ottima.

In assenza della assunzione del modello cosa sappiamo? Che la funzione di regressione è una media statistica!

$$\text{training set } T_n = \{\bar{X}_i, Y_i\}_{i=1}^n$$

Abbiamo a disposizione delle coppie nel training set, il modo ingenuo in cui potremmo pensare di rappresentare la r sapendo solo che è la media di $Y|X$ è la media campionaria.

La media campionaria di $Y|X$

NOTAZIONE

$$\hat{\pi}(\bar{X}_0; T_n) \triangleq \hat{\pi}_n(\bar{X}_0)$$

Quella a sinistra è una notazione pulita, una volta che noi non abbiamo la vera r ma la stiamo approssimando usando il dataset di training la r non è più solo funzione di X ma è anche funzione di T_n , per semplicità usiamo la forma a destra.

Principio 1: Sfruttare la LLN

Ricordiamo la Law of Large Numbers, se N è grande potremo effettivamente ottenere il risultato sperato.

$$\frac{1}{n} \sum_{i=1}^n Y_i \xrightarrow[n \rightarrow +\infty]{} \mathbb{E}[Y_0] \rightarrow \text{media a priori}$$

Questa è la media a priori, a noi serve la media a posteriori.

Come si fa?

$$\mu(2) = E[Y_0 | X_0 = 2]$$

Supponiamo di voler calcolare la media di Y dato che X è uguale a 2.

Intuitivamente possiamo estrarre da T_n tutte le label Y_i con $X_i = 2$ e calcolare la media aritmetica di queste label, lo vediamo in formula di seguito.

$$\hat{\mu}_n(2) = \frac{\sum_{i=1}^n Y_i \cdot \mathbb{I}[X_i = 2]}{\sum_{i=1}^n \mathbb{I}[X_i = 2]}$$

prendiamo solo le coppie con $X = 2$

INDICATORE = vale 1 se è verificato $X=2$ altrimenti 0

Per la legge dei grandi numeri si può dimostrare che questa espressione per n che va a infinito convergerebbe a $E[Y_0 | X_0 = 2]$.

Si solleva un problema.



Una variabile aleatoria discreta è una variabile che assume un numero finito di valori, ognuno con una determinata probabilità.



Una variabile aleatoria continua è particolare perché per ogni dato valore la probabilità che la variabile aleatoria continua lo assuma è 0 qualunque sia il valore scelto. Questo vuol dire che la definizione che abbiamo dato darebbe sempre come risultato 0 perché a numeratore ogni elemento della sommatoria varrebbe 0 essendo la probabilità di uno specifico valore.

Il fatto che l'evento a probabilità 0 può comunque succedere non ci aiuta perché noi ci basiamo per il ragionamento che stiamo facendo ora sulla Legge dei Grandi Numeri, ma se i numeri che effettivamente saranno conteggiati saranno pochissimi per un dataset di dimensioni ragionevoli allora è ovvio che non riusciamo ad avvalerci della LLN (faccio 1000 estrazioni e magari $X=2$ esce una sola volta, non mi aiuta).

Digressione sul concetto di probabilità 0.



Non è vero che un evento a probabilità 0 non può accadere, se un evento non può accadere significa che l'evento è impossibile.

Probabilità 0 significa che il rapporto tra le estrazioni che hanno portato a quel valore e il numero di estrazioni totale va a 0.

In altre parole l'evento si verifica ma con una frequenza assolutamente trascurabile.

Se ad esempio faccio k estrazioni e l'evento si verifica \sqrt{k} allora il limite per k che va a infinito è 0.

Principio 2: Località

NAIVE-KERNEL

Se non possiamo osservare proprio X_0 , almeno vogliamo essere vicini a X_0 .

Prendiamo i valori che sono non uguali a X_0 ma prossimi a esso, che fanno parte di un intorno di X_0 .

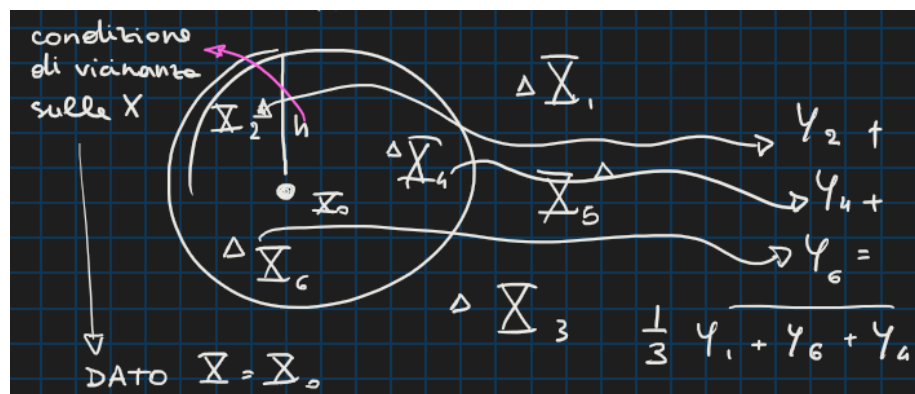
$$\hat{\pi}_n(X_0) = \frac{\sum_{i=1}^n Y_i \mathbb{I}[\|X_i - X_0\| \leq h]}{\sum_{i=1}^n \mathbb{I}[\|X_i - X_0\| \leq h]}$$

Potrebbero esserci problemi in cui le X sono alcune continue e alcune discrete (le discrete talvolta anche categoriche), ad esempio qualcosa di relativo al Covid dove abbiamo il dato Covid/non-Covid e il dato temperatura. Detto questo la ragione per la quale ci stiamo concentrando sul caso continuo è perché è il caso difficile.

Questo che abbiamo trascritto non è l'unico metodo che rispetta il principio di località, anzi, ce ne sono infiniti che si possono inventare per decidere "vicino" o "lontano".

Lo specifico metodo che abbiamo trascritto prende il nome di **Naive-kernel**, naive indica che è intuitivo, kernel indica una funzione che da una forma, qui al variare di h fa una sorta di finestra mobile, se si è all'interno di h valgo 1 altrimenti 0, il kernel più immediato è proprio dentro/fuori, o sei dentro o sei fuori dall'intorno, quindi naive, non sarebbe stato naive se ci si inventava le funzioni esponenziali pesate, le campane, le gaussiane ecc.

Rappresentazione grafica di Naive-kernel in un caso bidimensionale



Notiamo che, giustamente, la media è calcolata sulle label (Y_i) condizionatamente alle features (X_i)

Prendendo tutti quelli entro la distanza h da X_0 notiamo che sappiamo quanto sono vicini i punti selezionati MA non sappiamo quanti punti selezioniamo.

Esistono criteri alternativi.

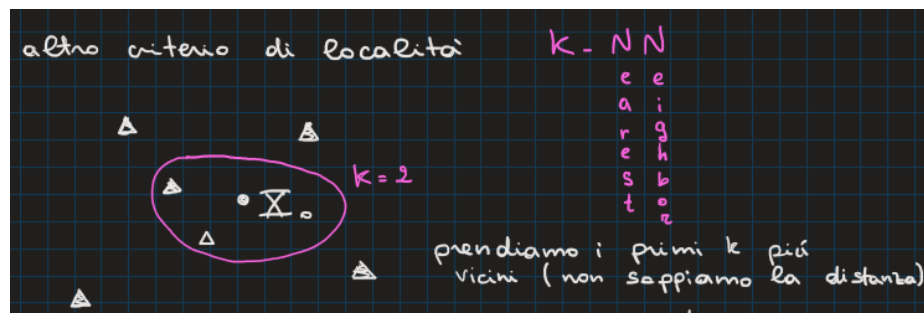
K-NEAREST NEIGHBOUR



Un criterio alternativo a Naive-kernel, che sceglie tutti i punti entro una certa distanza, il metodo K-nearest neighbour (K-NN) prendi i K più vicini.

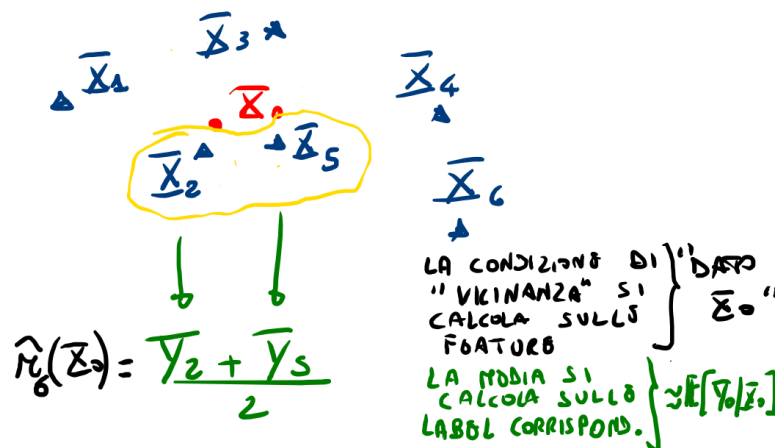
Con questo metodo sappiamo quanti punti selezioniamo, sempre K, ma non sappiamo effettivamente quanto sono vicini al nostro punto di interesse.

Rappresentazione grafica di K-nearest neighbour in un caso bidimensionale



Metodo K-NN

K=2



Naive-kernel vs. K-nearest neighbour

Per confrontare questi due metodi prima di tutto dobbiamo chiederci qual è il requisito che noi desideriamo: che per n che va a infinito la funzione di regressione stimata diventi sempre più vicina a quella vera.

Questo lo vedremo quando troveremo il modo di stimare i β nella regressione lineare, aumentando n ci avviciniamo ai β veri e ci avviciniamo alla funzione di regressione vera.

Noi vorremmo questo stesso comportamento ma senza l'assunzione sul modello.

LLN e Località per K-NN

Per ottenere questo risultato prima ci siamo ispirati alla LLN ma se con K-NN abbiamo sempre K osservazioni selezionate per calcolare la media anche se n va a infinito continuiamo ad avere K osservazioni e quindi con K-NN violiamo la Legge dei Grandi Numeri.

In merito alla località invece all'aumentare delle osservazioni la densità delle osservazioni nel piano aumenterà e quindi i K più vicini saranno sempre più vicini ad X_0 , quindi con K-NN sfruttiamo il Principio di Località.

LLN e Località per Naive-kernel

Se h è fisso per n che va a infinito il numero di osservazioni selezionate per la media aumenta, quindi la media viene calcolata su sempre più punti, con Naive-kernel sfruttiamo la Legge dei Grandi Numeri.

Ma se h è fisso indipendentemente da quanto aumentiamo la cardinalità del data-set i punti restano sempre entro la distanza h , non diventano più vicini, con Naive-kernel violiamo il Principio di Località, infatti in questo modo non si converge alla media statistica $E[Y|X = x_0]$ ma si converge invece alla media statistica $E[Y|X \in intorno]$.

Conclusioni

Possiamo quindi dire che questi due metodi sono complementari, duali.

Ora dobbiamo capire come si fa a garantire che se n cresce alla fine è possibile rispettare entrambi i principi.

Partiremo da Naive-kernel, che è facile da trattare anche a parole, poi ricaveremo K-NN partendo dal fatto che è il duale di Naive-kernel. Quindi su K-NN non faremo ragionamenti analitici.

Correggere Naive-kernel

L'unica possibilità è fare in modo che se n cresce l'intorno si deve rimpicciolire. Il problema è però che se lo riduciamo troppo dentro non ci sarà alcun punto.

Speriamo che esista un regime in cui h scende e l'intorno diventa sempre più piccolo ma comunque ci entrano infiniti punti (per n che va a infinito ovviamente).

Proviamo a risolvere questo problema con un esercizio.

Vogliamo che

$$h_n \xrightarrow{n \rightarrow +\infty} 0$$

Supponiamo un contesto a una dimensione, quindi lavoriamo con la variabile aleatoria continua $X \in \mathbb{R}$ della quale conosciamo la pdf $f(x)$.

Calcoliamo al crescere di n , rozzamente, quanti punti finiscono nell'intervallo di lunghezza h_n , quindi la condizione di nostro interesse è

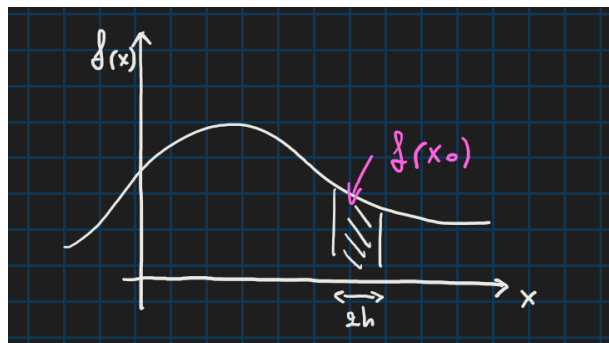
$$\|\bar{X}_0 - x_0\| \leq h$$

Usiamo la probabilità.

Dire che la distanza tra X_i ed x_0 è minore di h coincide con il dire che X_i appartiene all'intorno $[x_0 - h, x_0 + h]$.

$$\begin{aligned} P[\|\bar{X}_i - x_0\| < h] &= P[\bar{X}_i \in [x_0 - h, x_0 + h]] \\ &\quad \text{caso monodimensionale} \\ &= \int_{x_0 - h}^{x_0 + h} f(x) dx \approx 2h f(x_0) \approx \frac{n_p(x_0)}{n} \end{aligned}$$

La probabilità di una random variable continua si calcola con un integrale, se però h è molto piccolo l'integrale, che è l'area sottesa alla curva, può essere approssimato all'area di un rettangolo, che è base per altezza.



Rappresentazione grafica del significato di integrale

Noi vogliamo legare la probabilità al numero di punti, come si fa a legare la probabilità al numero di punti? Si usa il significato della probabilità, che ci suggerisce che la probabilità di un evento *assomiglia* al rapporto tra il numero di esperimenti che hanno dato un certo esito e il numero totale di esperimenti, per questo possiamo approssimare a $\frac{n_p(x_0)}{n}$ che significa numero di punti x_0 fratto numero di punti totale.

Ora ci chiediamo: il numero di punti che finisce nell'intorno, come si lega all'intorno? Moltiplichiamo entrambi i lati dell'ultima eguaglianza per n .

$$n_p(x_0) \approx 2 f(x_0) n h_n$$

Il numero di punti nell'intorno "va" come $n * h * f(x_0)$.

Da adesso h lo chiamiamo h_n .

E ora la domanda diventa, come scegliamo h_n in modo che il numero di punti che finisce nell'intorno vada ad infinito?

Visto che abbiamo che il numero di punti nell'intorno è uguale $n * h_n$ per una costante ($2f(x_0)$) bisogna fare in modo che

$$n h_n \rightarrow \infty$$

Vogliamo che $n h_n \rightarrow \infty$.

garantisce che $n_p(x_0) \rightarrow +\infty$ cioè LLN

E per quello che abbiamo detto sul fatto che l'intorno deve rimpicciolirsi perché altrimenti violiamo la Località sappiamo anche che

$h_n \rightarrow 0$ garantisce località

Vogliamo che $h_n \rightarrow 0$.

E queste sono le due condizioni di nostro interesse.

Dobbiamo fare in modo che entrambe queste condizioni siano verificate insieme, un valore possibile di h_n per il quale ciò accade è:

$$h_n = \frac{1}{\sqrt{n}}$$

Ma ci sono ovviamente infiniti possibili valori di h_n per i quali le due condizioni sono verificate.

Correggere K-NN

La legge su K si fa partendo dal presupposto che K_n è l'inverso di h_n quindi

$$K_n = \frac{1}{h_n}$$

Per rispettare la Legge dei Grandi Numeri vogliamo che $K_n \rightarrow \infty$ che è rispettato visto che $h_n \rightarrow 0$ e sta a denominatore ed è l'inverso della prima condizione per Naive-kernel.

Inoltre vogliamo che $\frac{K_n}{n} \rightarrow 0$, che è l'inverso della seconda condizione per Naive-kernel, cioè i vicini devono crescere meno che n altrimenti non è più locale.

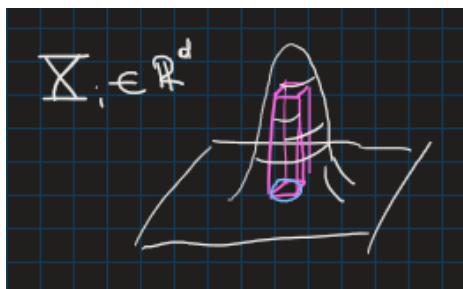
Conclusioni

Tutti i ragionamenti fino ad ora sono stati fatti per il caso monodimensionale, vedremo anche quello a più dimensioni, essenzialmente se nell'integrale in questo caso abbiamo approssimato ad un rettangolo con più dimensioni diventa un parallelepipedo e poi un iper-cubo e quindi invece di base per altezza diventa con h al quadrato, alla terza e così via.

All'aumentare delle dimensioni il numero di punti necessario (numero di osservazioni) aumenta, si parla di **Curse of dimensionality** (maledizione della dimensionalità).

Naive-kernel e K-NN in caso multidimensionale - 12/11

Ci siamo chiesti cosa succede se passiamo dall'esempio monodimensionale ad uno multidimensionale.



Integrale di f_X con $X \in R^2$

Se siamo in caso bidimensionale possiamo approssimare l'integrale con il volume del parallelepipedo di base $h * h$ e altezza $f(x_0)$.

Il discorso sulla probabilità diventa quindi:

$$\mathbb{P}[\|X - X_0\| < h] \approx \frac{n_p(X_0)}{n} \approx h^d f(X_0) \quad \rightarrow \text{pdf multivariata}$$

E quindi

$$n_p(X_0) \approx n h^d f(X_0)$$

La norma ora la stiamo facendo in due dimensioni però la norma quadra minore di h quadro è tipo una circonferenza.

Per tenere conto di questa approssimazione bisogna aggiungere una costante c_d che cambia a seconda di cosa sto calcolando (che potrebbe ad esempio essere πr^2) che non ci interessa troppo con precisione, l'importante una volta messa la costante è garantire che $n_p(x_0) \rightarrow \infty$.

$$\mathbb{P}[\|\bar{X} - X_0\| \leq h] \approx \frac{n_p(x_0)}{n} \approx c_d h^d f(x_0) \quad \text{p.d.f. multivariata}$$

$$n_p(x_0) \approx c_d n h^d f(x_0)$$

Per **Naive-kernel** le condizioni di nostro interesse in caso multidimensionale con $n \rightarrow \infty$ sono:

$$n h_n^d \rightarrow \infty$$

$$h_n \rightarrow 0$$

Notiamo che c'è una **dipendenza esponenziale**, se h_n è molto piccolo (che è il nostro caso perché noi vogliamo tenda a 0) allora h_n^2 è ancora più piccolo e h_n^{10} lo sarà molto di più e quindi visto che h_n^d sarà significativamente più piccolo di h_n allora sarà necessario per far sì che $n h_n^d \rightarrow \infty$ un valore di n molto più grande rispetto a quanto sarebbe bastato nel caso monodimensionale.

Al crescere della dimensionalità esplode il numero di osservazioni che sono necessarie per risolvere i problemi di apprendimento, questa è nota come **Curse of Dimensionality**. Intuitivamente 9 punti su una retta sono molto più densi di 9 punti su un piano, lo stesso numero di osservazioni al crescere della dimensionalità perde di finezza, **per avere la stessa finezza di K osservazioni in monodimensionale sono necessarie K^d osservazioni in d-dimensionale**.

Per **K-NN** basta ragionare per estensione del ragionamento fatto per Naive-kernel, senza fare conti che volendo si potrebbero fare ma che sono molto più sofisticati quindi evitiamo. I conti diventano complicati perché dovremmo capire quando cambia K i K più vicini quanto si avvicinano a x_0 .

Per K-NN le condizioni di nostro interesse in caso multidimensionale con $n \rightarrow \infty$ sono:

$$\begin{array}{c} \text{K-NN} \\ K_n \xrightarrow{n \rightarrow +\infty} +\infty \\ \frac{K_n}{n} \xrightarrow{n \rightarrow +\infty} 0 \end{array}$$

La prima condizione significa che il numero di vicini deve andare all'infinito per garantire la LLN.

La seconda condizione significa che il numero di punti nell'intorno per n infinito deve andare a 0 (facile da capire se si riguarda l'equazione $\frac{n_p(x_0)}{n} \simeq c_d h_n f(x_0)$).

Errore nella regressione non parametrica

Partiamo dal teorema sull'errore della funzione di regressione.

Da Matta ci è stato presentato come una sorta di esercizio "capire se l'errore quadratico di stima con il metodo che usa \hat{r} stimato a partire dal training set è legato all'errore quadratico di stima che si ottiene

con la vera funzione di regressione r^*

Si ricorda che \hat{r} è aleatoria in quanto dipende dal Training set che può essere ricampionato, tuttavia se lo fissiamo non c'è più aleatorietà dovuta a T_n e l'unica aleatorietà è ora dipesa da Y_0 e X_0 , non c'è più aleatorietà dovuta alla fase di training ma solo dovuta alla fase di predizione, si dice che "congeliamo" il data set.

$$\mathbb{E}[(Y_0 - \hat{r}_n(X_0))^2 | T_n]$$

Detto questo continuiamo con dei ragionamenti a partire da:

$$(*) \Rightarrow \mathbb{E}\{[\hat{r}(X_0) - Y_0]^2\} = \text{MMSE} + \mathbb{E}\{[\hat{r}(X_0) - r(X_0)]^2\}$$

↳ errore col rischio empirico
↳ distanza quadratica tra \hat{r} e r
termine di penalità

L'MMSE è un termine intrinseco ed ineliminabile.

$$\mathbb{E}[(Y_0 - \hat{r}_n(X_0))^2] = \text{MMSE} + \mathbb{E}[(\hat{r}(X_0) - r(X_0))^2]$$

non si tocca
un errore intrinseco del mio modello non lo posso scendere

Come detto abbiamo congelato il Training set, altrimenti non ci sarebbe da valutare solo l'aleatorietà di predizione ma anche quella di training.

$$\mathbb{E}[(Y_0 - \hat{r}_n(X_0))^2 | T_n] = \text{MMSE} + \mathbb{E}[(\hat{r}(X_0) - r(X_0))^2 | T_n]$$

predizione
scendere

 $\hat{r}(X_0; T_n)$
 $n \rightarrow \infty \rightarrow 0$
strong consistency
tutte le realizzazioni vanno a zero



Usando la Tower Property possiamo mediare anche per le realizzazione del Training Set, possiamo così togliere il condizionamento.

questa relazione vale anche (Per la tower property)

$$\mathbb{E}[(Y_0 - \hat{r}_n(X_0))^2] = \text{MMSE} + \mathbb{E}[(\hat{r}(X_0) - r(X_0))^2]$$

si media attraverso le training set

 $n \rightarrow \infty \rightarrow 0$
weak consistency
mediando tutte le relazioni del training set va a zero

Vorremmo che per n a infinito il secondo termine di errore vada a 0, per la prima relazione si parla di Strong Consistency, per la seconda Weak Consistency.

Le due forme di consistenza non si implicano tra loro, il che porta a chiederci come farebbe ad andare a 0 in media la quantità se tutte le singole realizzazioni non vanno a 0, matematicamente è troppo complesso, noi ci limitiamo a capire che il tendere a 0 per n infinito è diverso per le due formule. Una è weak consistence e l'altra è strong consistence, non si implicano tra di loro.

Ci interessa tutto questo perché le condizioni (i limiti) che abbiamo detto prima per Naive-kernel e K-NN implicano la Weak Consistency e quindi ci garantiscono di tendere all'errore ottimo.

Regressione non parametrica in Matlab

Scriviamo una funzione per la regressione non parametrica, è buono se il file si chiama allo stesso modo.

```
function [fkNN, fNKer, f] = NonparametricRegression(k, h, n, a)

% Training set generation
e = randn(1, n); x = a * rand(1, n); y = sin(2*pi*x) + e;
x0 = linspace(0, a, 200);
r = sin(2*pi*x0);

for ii=1:length(x0)

    % k-NN
    d = abs(x-x0(ii));
    [~, ind] = sort(d);
    rkNN(ii) = mean( y(ind(1:k)) );

    % naive kernel
    local = ( abs(x-x0(ii)) <= h )
    rNKer(ii) = sum( y.*local ) / sum(local);
end

plot(x0, rkNN, 'b', x0, rNKer, 'g', x0, r, 'r', 'LineWidth',2);
xlabel('$x_0$', 'FontSize',20)
ylabel('Regression functions', 'FontSize',20)
leg=legend('k-NN', 'Naïve Kernel', 'Optimal regression function')
set(leg, 'fontsize', 20)
```

Gli argomenti sono il k di K-NN, la h di Naive-kernel, n che è il numero di campioni ed a che è il massimo valore che possono assumere le X .

```
function [fkNN, fNKer, f] = NonparametricRegression(k, h, n, a)
```

Generiamo il training set.

```
% genera un vettore 1Xn di valori casuali estratti da una normale standard
e = randn(1, n); % e è l'errore epsilon della forma  $Y = r(X) + \text{epsilon}$ 

% genera un valore tra 0 e a
x = a * rand(1, n);

% genera le y
y = sin(2*pi*x) + e;
% per assicurarci che questa sia regressione ci serve controllare che
% la media di e dato x faccia 0, in questo caso è ovvio perché sono indipendenti
% ed e viene da una Gaussiana a media 0

% l'altra proprietà importante è che i campioni X del training set sono stati
% generati in maniera indipendente
```

Ora vogliamo tracciare la funzione di regressione, quindi ci serve valutarla in un numero molto grande di punti (come abbiamo fatto in R con MLE)

```
% genera punti equidistanziati nei quali valutare la funzione di regressione
x0 = linspace(0, a, 200); % quindi non stiamo simulando il sistema o facendo predizione
% vogliamo disegnare la funzione, x0 sono 200 punti equidistanziati da 0 ad a
% ora quando usiamo Naive-kernel o K-NN disegniamo anche il risultato
```

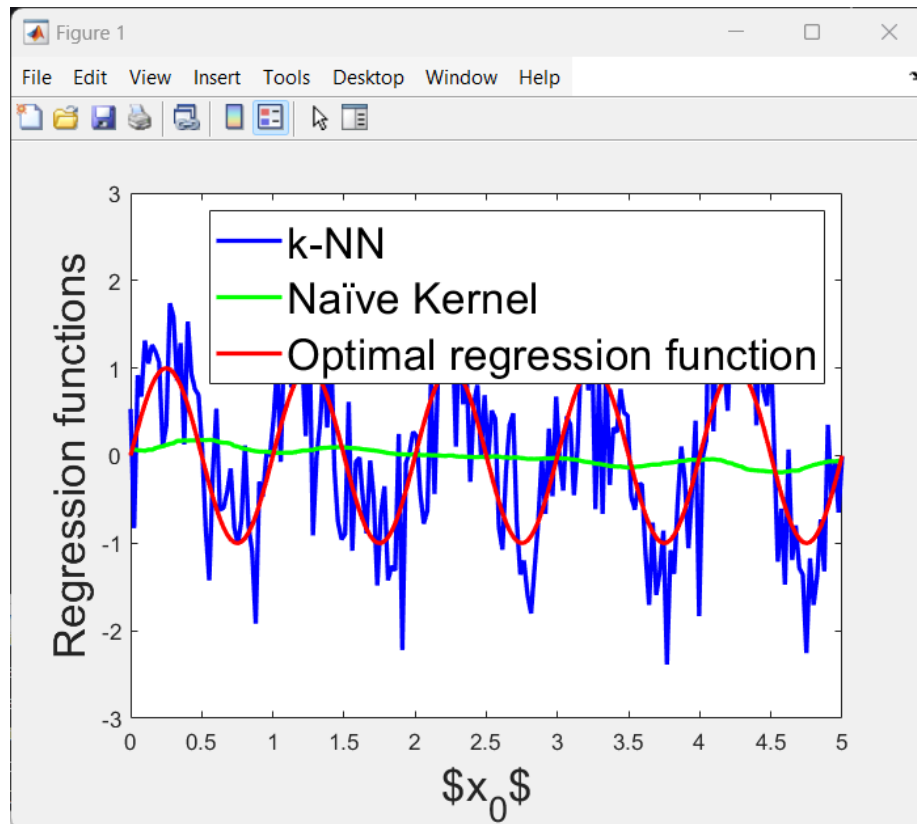
```
r = sin(2*pi*x0); % in r abbiamo la forma della funzione di regressione
% perché applicando una funzione ad un vettore, questa viene calcolata in tutti
% i punti del vettore
```

Per ogni punto di x_0 vogliamo **calcolare la funzione di regressione** del K-NN e quella del Naive-kernel.

```
% k-NN
d = abs(x-x0(ii));
[~, ind] = sort(d);
rkNN(ii) = mean( y(ind(1:k)) );
```

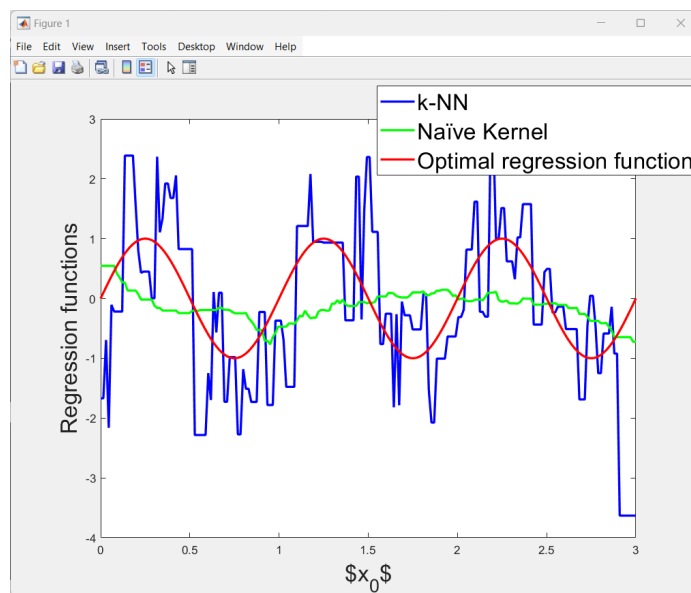
```
% naive kernel
local = ( abs(x-x0(ii)) <= h )
rNKer(ii) = sum( y.*local ) / sum(local);
```

Il plotting porta, per uno specifico input (2, 2, 1000, 5), a questo risultato:



Comportamento al variare dei parametri

Output per `NonparametricRegression(1,0.5,100,3)`:



Il risultato sembra insoddisfacente.

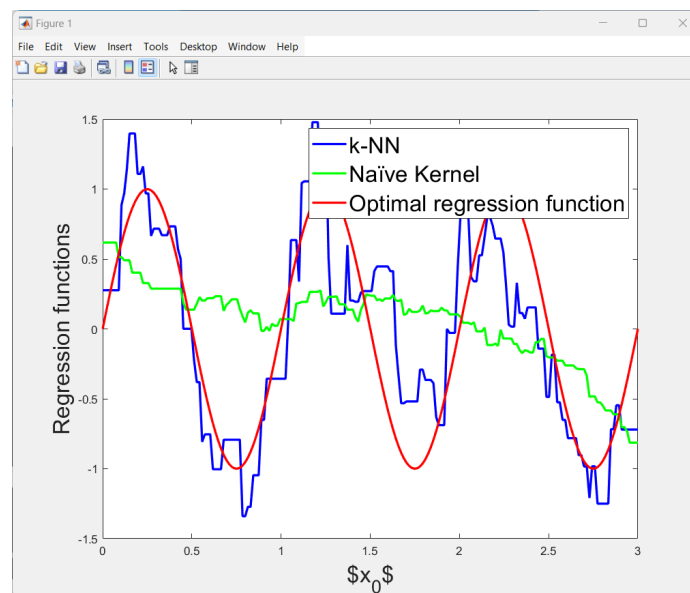
K-NN

Proviamo a migliorare il **Nearest neighbour**, un solo vicino evidentemente non è sufficiente. Con n fisso non ha senso usare la legge che lega k ad n come $k = \sqrt{n}$ perché quella è una legge asintotica, ora n è fisso quindi non ci porta a nulla usare la legge asintotica.

Non significa neanche nulla dire $7\sqrt{n}$ o $100\sqrt{n}$ o $1000\sqrt{n}$. Nei teoremi asintotici in Machine Learning c'è sempre la famosa costante, se davanti a radical n metto 10^8 in un problema pratico poi conta solo 10^8 e il resto non lo vedo, questo non toglie pregio alle analisi asintotiche MA bisogna sapere che sono asintotiche.

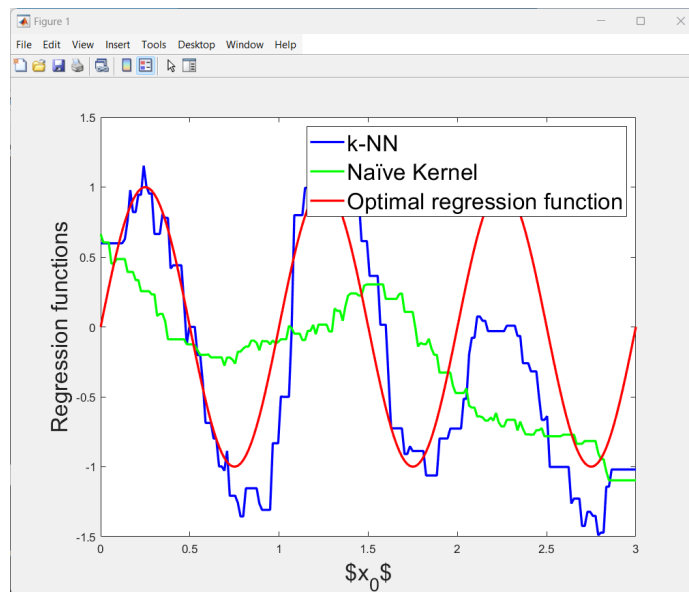
Questo vuol dire che per scegliere K si deve fare una fase di validazione (tuning), si sceglie a tentativi essenzialmente.

Aumentando i vicini a 5 otteniamo:

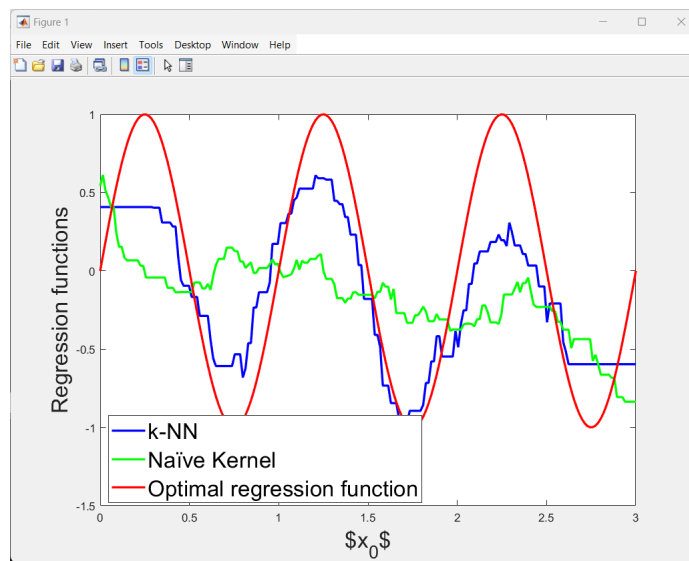


Che sembra effettivamente più fedele.

Con 10:

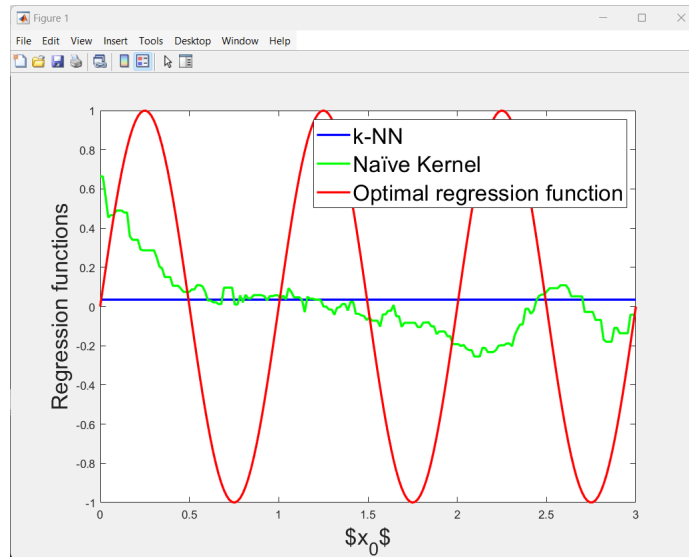


Con 20:



Notiamo che se cresce troppo il numero di vicini la K-NN perde a partire dai bordi la capacità di predizione, si chiama effetto di bordo, i valori ai bordi non hanno vicino da un lato e quindi si appiattiscono.

Prendendo tutti e 100 i campioni come vicini:

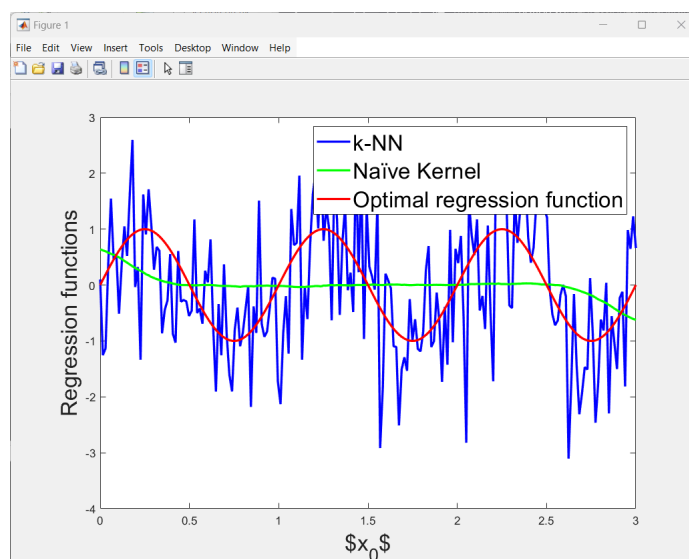


Diventa una retta che è la media di tutti i valori, non esce proprio 0 perché è la media di Y senza il "dato X " (solo $E[Y]$ invece di $E[Y|X]$), non è proprio 0 perché con un numero finito di campioni comunque non è perfetta la stima.

Prendendo tutti i campioni violo la località.

Invece quando prendiamo un solo campione come vicino rispettiamo la località ma non rispettiamo la LLN e quindi le stime ballano molto.

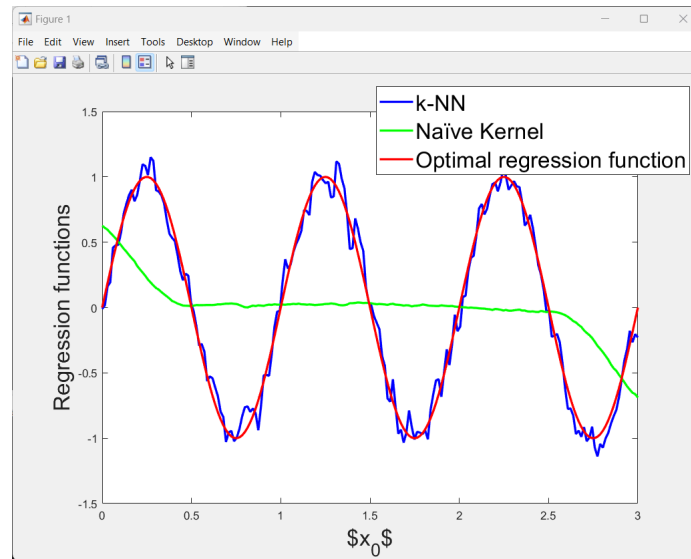
Se portiamo il numero di campioni a 10000 e consideriamo un solo vicino, `NonparametricRegression(1,0.5,10000,3)`, notiamo che balliamo ancora molto ma balliamo meglio intorno al valore vero.



Aumentando moltissimo i campioni comunque balliamo molto perché abbiamo migliorato tanto la località facendo crescere n ma non abbiamo migliorato la LLN perché consideriamo sempre un solo vicino.

Questo problema lo risolviamo aumentando il numero di vicini in maniera opportuna, con tentativi vediamo che con 100 vicini le cose iniziano a funzionare.

`NonparametricRegression(100,0.5,10000,3):`



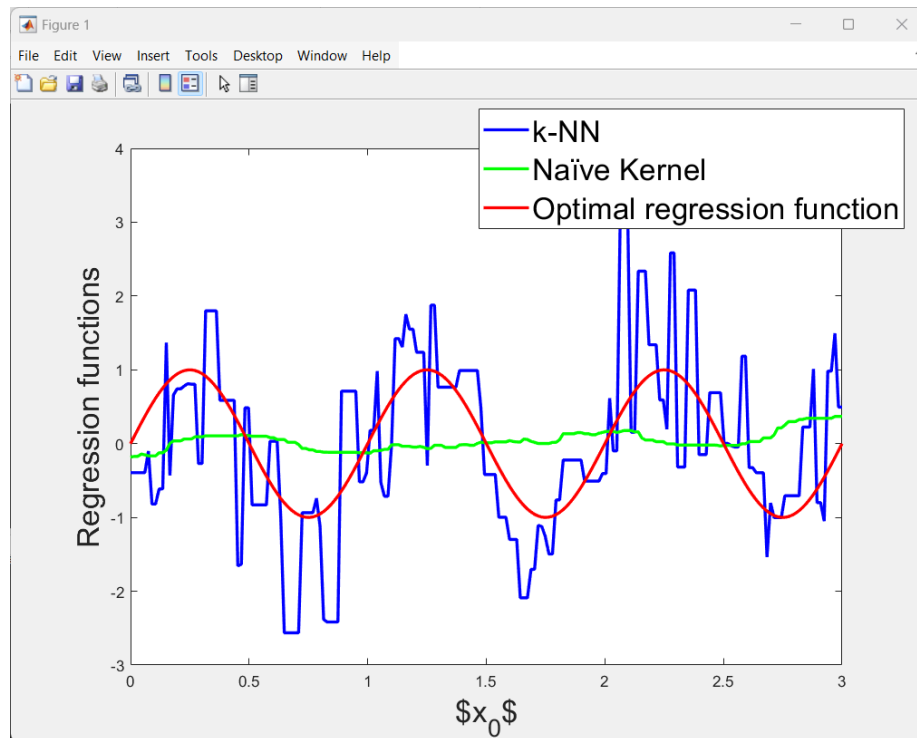
Sembra un risultato soddisfacente.

Il senso fisico di dire K deve essere grande ma non troppo grande è che violiamo la località altrimenti.

Naive-kernel

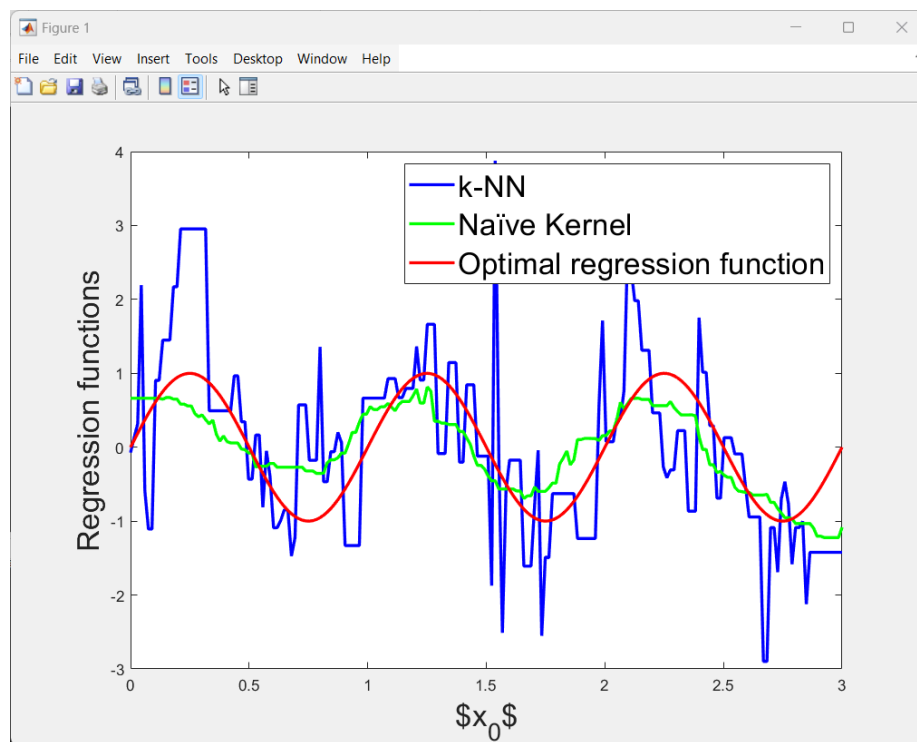
X è un uniforme tra 0 e 3 quindi i valori di h devono essere commisurati a questo 3.

Se h lo faccio 1 su 3 violo la Località, il valore è troppo grande, infatti vediamo con `NonparametricRegression(1,1,100,3)` che la curva verde balla poco ma non c'entra niente con la rossa.

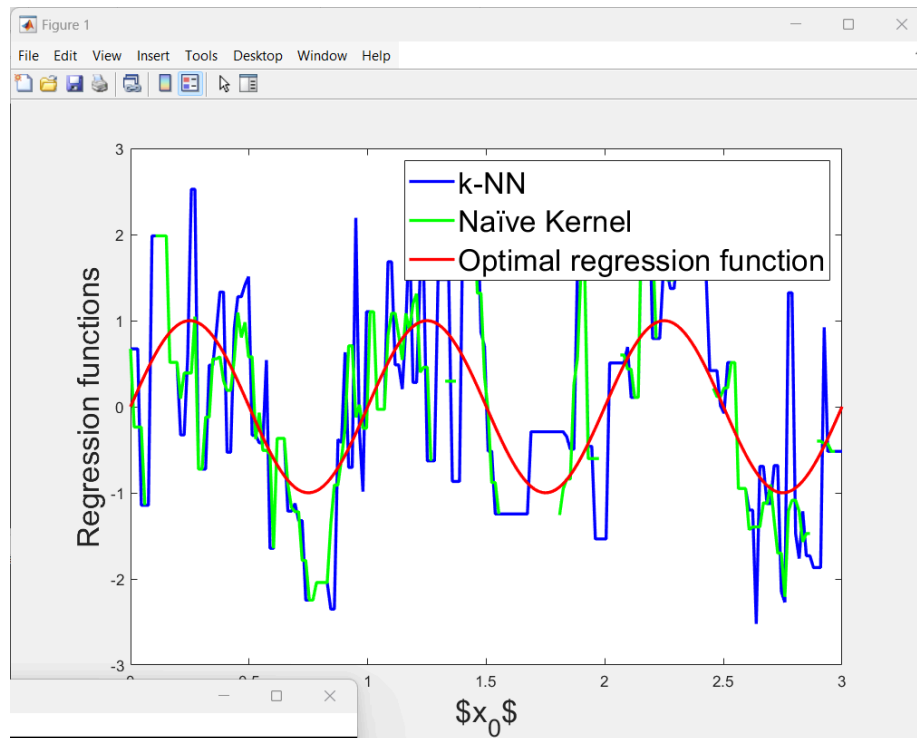


Aumentando h fino a 3 arriviamo ad ottenere la media, è il caso corrispondente a $K = n$.

Se h lo facciamo più piccolo tendiamo alla rossa ma visto che i campioni sono pochi diventa più rumoroso, perdiamo punti negli interni e la legge dei grandi numeri non funziona.



Con $h = 0.3$

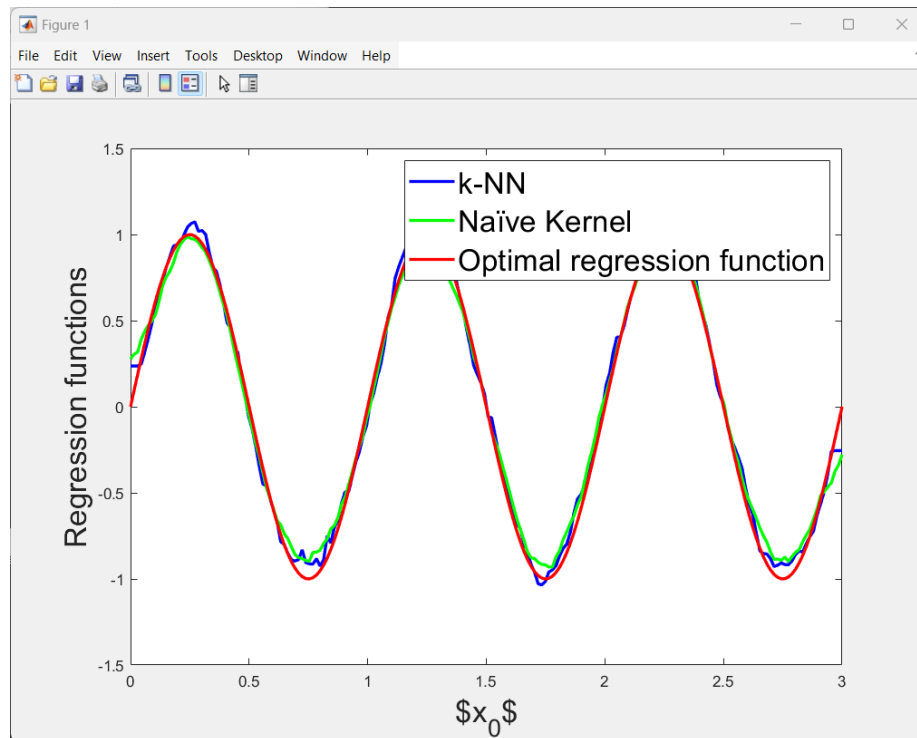


Con $h = 0.03$

Con un h troppo piccolo notiamo oltre al forte rumore addirittura dei buchi che sono i punti dove non sono stati trovati vicini, con un intorno troppo piccolo è quasi come tornare al caso non funzionante di prendere invece che l'intorno proprio lo specifico punto.

Con $h = 0.1$ e 10000 punti otteniamo un buon risultato. Abbiamo finito.

`NonparametricRegression(300,0.1,10000,3)`



Assegno

Realizzare questo programma, usarlo per fare delle simulazioni Monte Carlo per calcolare le medie che abbiamo scritto alla lavagna, la X_0 non dovrà più essere...

Possiamo anche modificare il programma a piacimento, potremmo scrivere un altro programma in cui calcoliamo solo le X_0 che verranno realizzate in fase di predizione. Cioè invece di fare il plot a noi può interessare fare specifiche predizioni su certi valori.

Simulazione che vuol dire, facciamo la Weak Consistency, cioè non dato il Training set, quindi nella simulazione Monte Carlo per ogni ciclo, per ogni iterazione Monte Carlo dobbiamo rigenerare un Training set, dobbiamo rigenerare un valore X_0 di predizione, Y_0 di predizione, calcolare Naive-kernel e K-NN, conservare l'errore quadratico e in uscita al ciclo monte carlo calcolare la media. In questo modo mediamo sia sul Training set che su X_0 e Y_0 .

NON separare le fasi, non fissare il training set e poi fare il programma e poi fare un altro programma dopo aver fissato un altro training set.

Il vantaggio delle simulazioni Monte Carlo è proprio che indipendentemente da quante sono le variabili se generiamo tutto insieme ogni volta a ogni iterazione mediamo rispetto a TUTTO.

A ogni iterazione generiamo le nostre cose aleatorie e a quei valori applichiamo il nostro stimatore.

Finito tutto rifacciamo il calcolo con diversi valori di n , scelti come? Spaziati logaritmicamente, 10, 1000, 10000, una decina di punti da 10 a 10000.

Un bel plot alla fine che ci fa vedere l'errore come varia.

In genere le leggi tipo radice funzionano (per il tuning che dovremo fare) ma non è detto siano le migliori.

Svolgimento

MATLAB

```
function MonteCarloSimulation(k, h, a, mc_iterations, n_values)
    % k: Numero di vicini per k-NN
    % h: Banda per il Naive-kernel
    % a: Limite superiore per il range dei dati
    % mc_iterations: Numero di iterazioni Monte Carlo
    % n_values: Vettore dei valori di n (dimensioni del training set)

    mse_kNN = zeros(size(n_values));
    mse_NKer = zeros(size(n_values));

    n_values = round(n_values);

    for idx = 1:length(n_values)
        n = n_values(idx);
        error_kNN = zeros(1, mc_iterations);
        error_NKer = zeros(1, mc_iterations);

        for mc = 1:mc_iterations
            % Generazione del training set
            e = randn(1, n);
            x = a * rand(1, n);
            y = sin(2 * pi * x) + e;

            % Generazione del punto di predizione
            x0 = a * rand();
            y0 = sin(2 * pi * x0) + randn();

            % k-NN
            d = abs(x - x0);
            [~, ind] = sort(d);
            y_kNN = mean(y(ind(1:k)));

            % Naive kernel
            local = (abs(x - x0) <= h);
            y_NKer = sum(y .* local) / sum(local);

            % Calcolo dell'errore quadratico
            error_kNN(mc) = (y0 - y_kNN)^2;
            error_NKer(mc) = (y0 - y_NKer)^2;
        end
    end
```



```

    % Media degli errori quadratici
    mse_kNN(idx) = mean(error_kNN);
    mse_NKer(idx) = mean(error_NKer);
end

% Plot dei risultati
figure;
loglog(n_values, mse_kNN, '-o', 'LineWidth', 2);
hold on;
loglog(n_values, mse_NKer, '-s', 'LineWidth', 2);
xlabel('Training set size (n)', 'FontSize', 14);
ylabel('Mean Squared Error (MSE)', 'FontSize', 14);
legend('k-NN', 'Naive Kernel', 'FontSize', 14);
grid on;
title('MSE vs Training set size', 'FontSize', 16);
end

```

```

k = 5;      % Numero di vicini
h = 0.5;    % Banda per il Naive Kernel
a = 5;      % Range dei dati
mc_iterations = 1000; % Iterazioni Monte Carlo
n_values = logspace(1, 5, 12); % n distribuiti logarithmicamente tra 10 e 10.000

MonteCarloSimulation(k, h, a, mc_iterations, n_values);

```