Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in

STATISTICA PER L'ECONOMIA E L'IMPRESA

RELAZIONE FINALE

Metodi di apprendimento automatico contro metodi statistici per la previsione di serie storiche

Relatore: Prof.ssa Luisa Bisaglia
Dipartimento di Scienze Statistiche

Laureando: Antonio Mattesco
Matricola N 2002142

Anno Accademico 2022/2023

Indice

Introduzione

La previsione delle serie storiche è uno dei temi di ricerca più attivi. I metodi di apprendimento automatico sono sempre più adottati per risolvere problemi predittivi. Tuttavia, in un recente lavoro di Makridakis et al. 2018, è stato dimostrato che questi approcci presentano sistematicamente una performance predittiva inferiore rispetto ai metodi statistici e che la maggior parte dei metodi di apprendimento automatico non riesce a prevedere meglio di un semplice modello *random walk stagionale*. In questo lavoro, vengono contraddetti questi risultati replicando l'esperimento presentato nell'articolo *"Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters"* (Cerqueira, Torgo e Soares 2019) ed effettuando, in seguito, le stesse analisi su serie storiche differenti, per mostrare che questi sono validi solo sotto un campione di dimensione ridotta. Infatti, lo studio presentato da Makridakis e i suoi collaboratori si basava su 1045 serie storiche mensili utilizzate nella competizione M3 (Makridakis e Hibon 2000) il cui numero medio, minimo e massimo di osservazioni erano rispettivamente 118, 66 e 144. L'ipotesi è che queste serie siano troppo corte perché un modello di apprendimento automatico possa adattarsi correttamente. Quando la dimensione dei dati è piccola, il campione potrebbe non essere rappresentativo del processo che genera le serie sottostanti.

I risultati suggeriscono che i metodi di apprendimento automatico migliorano la loro performance predittiva man mano che la numerosità campionaria cresce.

L'obbiettivo finale di questo lavoro è quello di confrontare i metodi di apprendimento automatico con quelli statistici per la previsione delle serie storiche ponendo particolare attenzione alla loro dimensione.

La relazione è strutturata come segue: nel capitolo 1 vengono descritti i modelli utilizzati, nel capitolo 2 si presenta la metodologia adottata, i risultati della replicazione dell'esperimento vengono riportati nel capitolo 3 e infine nel capitolo 4 si discutono i risultati dell'analisi svolta su un altro insieme di dati.

Capitolo 1

I Modelli utilizzati

In letteratura sono stati proposti diversi modelli per l'analisi delle serie storiche. Questi non sono stati concepiti solo per prevedere il comportamento futuro delle serie, ma anche per aiutare a comprendere la struttura sottostante dei dati. Nel presente capitolo vengono presentati 5 modelli che diremo appartenere al gruppo dei **metodi statistici**: *SARIMA*, *THETA*, *Naive stagionale*, *ETS* (Exponential Smoothing State Space) e *TBATS* (Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend, and Seasonal components).
I 5 modelli successivi invece appartengono ai **metodi di apprendimento automatico** e sono: *Rule-Based Tree*, *Random Forset*, *MARS* (Multivariate Adaptive Regression Splines), *Gaussian Process* e *Generalized linear models*. Dal punto di vista dell'apprendimento automatico, la previsione di serie temporali è solitamente basato, su un modello AR(p) in accordo col *Teorema di Takens* (Takens 1981). L'idea è che le informazioni sulla dinamica del sistema siano contenute nella serie temporale stessa. Queste informazioni possono essere recuperate utilizzando un'operazione di incorporazione che coinvolge il ritardo temporale. Quindi si crea un vettore p-dimensionale, utilizzando p ritardi temporali, che costituisce la matri-

ce dei regressori. Formalmente si può scrivere che l'i-esima variabile esplicativa, relativa all'i-esimo valore y_i osservato della serie storica, è data da:

$$x_i = [y_{i-1}, ..., y_{i-p}]$$

In altre parole, il principio alla base di questo approccio è quello di modellare la distribuzione condizionale del valore della serie temporale dati i suoi p valori passati. Seguendo questa formulazione, possiamo ricorrere agli algoritmi di regressione per risolvere il compito predittivo. Nell'esperimento si pone $p = 10$, tuttavia questo parametro può essere ottimizzato seguendo Kenne et al. 1992.

1.1 Modelli SARIMA

I modelli *ARIMA stagionali* (Box e Jenkins 1976), o *SARIMA*, permettono di modellare congiuntamente le componenti stocastiche di trend e stagionalità. L'idea alla base è che il processo deve poter descrivere due tipi di relazioni all'interno della serie osservata: la correlazione tra valori consecutivi della serie e la correlazione tra osservazioni che distano tra loro un multiplo del periodo.

Un processo $ARIMA(p, d, q)(P, D, Q)_s$ stagionale è dato da:

$$\Phi(B^S)\phi(B)(1 - B^S)^D(1 - B)^d y_t = c + \Theta(B^S)\theta(B)\epsilon_t$$

dove ϵ_t è un processo *white noise* con media zero e varianza σ^2, B è l'operatore ritardo, $\phi(z)$ e $\theta(z)$ sono polinomi rispettivamente di ordine p e q, $\Phi(B^s)$ e $\Theta(B^s)$ sono polinomi di ordini P e Q, s è il periodo stagionale. I polinomi che si riferiscono agli ordini (p, d, q) hanno la stessa valenza dei modelli $ARIMA$ e sono dunque rispettivamente

gli operatori non stagionali a media mobile, di differenza non stagionale e autoregressivo. Invece i polinomi relativi agli ordini (P, D, Q) sono operatori stagionali a media mobile, di differenza e autoregressivo. Un *SARIMA* è stazionario se le radici dell'equazione $\Phi(B^S)\phi(B) = 0$ sono tutte in modulo maggiori di uno. Dato un processo stazionario qualsiasi è possibile calcolare in modo univoco la sua funzione di autocovarianza, non vale sempre il viceversa a meno che non sia rispettata la condizione di invertibilità. L'invertibilità riguarda la possibilità di esprimere un processo come una funzione delle variabili casuali del passato; il processo è invertibile se le radici dell'equazione $\Theta(B^S)\theta(B) = 0$ sono tutte in modulo maggiori di uno.

L'obbiettivo è quello di identificare i valori di p, P, d, D, q, Q e stimare i relativi parametri. Da un pusto di vista pretico ciò può essere ottenuto con la funzione *auto.arima()* del pacchetto *forecast* (R. Hyndman, Koehler et al. 2008) di R che restituisce il miglior modello in base acirteri di informazione automatica AICc, AIC e BIC. Per prima cosa si determinano i valori d e D rispettivamente in base ai test KPSS (Kwiatkowski et al. 1992) e Canova-Hansen (Canova e Hansen 1995). La funzione segue l'algoritmo *step-wise* proposto da R. J. Hyndman e Khandakar (2008) che, a parte in situazioni particolari dettagliate dagli autori, identifica un modello valido che sarà poi usato per produrre previsioni.

Auto.arima() si assicura che il modello adattato sia invertibile e stazionario imponendo vincoli sui parametri durante l'ottimizzazione: in particolare, quando il polinomio AR è vicino a essere non stazionario o quando il polinomio MA è vicino a essere non invertibile, il modello viene respinto impostando un valore infinito per l'AIC

relativo a quel modello.

1.2 Metodo THETA

Il modello Theta (Assimakopoulos e Nikolopoulos 2000a) propone un approccio diverso rispetto alle tecniche basate sulla decomposizione di trend-ciclicità, stagionalità e errore: si occupa di decomporre la serie destagionalizzata in due componenti di breve e lungo termine. Le due componenti vengono estrapolate separatamente, poi si effettuano le previsioni e infine vengono combinate per ottenere un valore finale. La combinazione, sotto determinate circostanze, migliora l'accuratezza previsiva.

Il modello è basato sull'idea di modificare la curvatura della serie storica grazie ad un coefficiente θ che viene applicato direttamente sulla differenza seconda della serie:

$$X''_{new}(\theta) = \theta X''_{data}$$

dove $X''_{data} = (1 - B)^2 X_t$ al tempo t. Minore è il valore di θ, maggiore è il grado di deflazione, fino al raggiungimento del caso limite di $\theta = 0$ per cui la serie è trasformata in una retta di regressione: infatti ponendo la derivata seconda di una funzione pari a zero, in ogni punto, si ottiene una retta. La progressiva diminuzione delle fluttuazioni diminuisce la differenze assolute tra i termini successivi della serie derivata ed è correlata all'emergere di tendenze di lungo periodo nei dati. Viceversa se la curvatura viene incrementata, la serie sarà dilatata proporzionalmente all'aumentare del parametro.

La procedura per la stima del modello si compone delle seguenti fasi:

Fase 1. Viene testata la stagionalità della serie utilizzando il t-test. Il t-test sulla funzione di autocorrelazione con un ritardo di un anno (cioè per una serie temporale mensile con 12 osservazioni e per una serie temporale trimestrale con 4 osservazioni) viene eseguito confrontando i valori della funzione con un intervallo di confidenza basato sulla distribuzione t di Student: in particolare si calcola

$$t^* = \frac{r}{\frac{\sqrt{(1-r^2}}{n}}$$

dove r rappresenta il valore dell'autocorrelazione per il ritardo e la si confronta con il quantile 0.9 della t di student. Se il valore della statistica del test t^* supera il valore critico, si può concludere che la correlazione tra le osservazioni adiacenti con ritardo di un anno è statisticamente significativa

Fase 2. Qualora fosse stata rilevato un comportamento stagionale, la serie viene destagionalizzata.

Fase 3. La serie viene decomposta in due θ-*lines*, la retta di regressione ($\theta = 0$) e la linea con $\theta = 2$. Porre il coefficente pari a 2 deriva dal bisogno di bilanciare la semplificazione derivante dall'uso della retta di regressione, aumentando l'irregolarità della serie storica e amplificando le tendenze più recenti.

Fase 4. La retta di regressione viene estrapolata col metodo dei minimi quadrati, mentre per la seconda linea si usa il lisciamento esponenziale.

Fase 5. Si effettua la previsione con entrambe le linee ottenendo $L(\theta = 0)$ per la prima e $L(\theta = 2)$ per la seconda. Per la stima finale si combinano con pesi uguali le due previsioni:

$$\hat{y} = \frac{1}{2}(L(\theta = 0) + L(\theta = 2))$$

Fase6. Si aggiunge la componente stagionale alla previsione trovata al punto precedente.

La prima θ-*line* ($\theta = 0$) descrive la serie storica attraverso un trend lineare; la seconda, raddoppia la curvatura ingrandendo il comportamento a breve termine. Nel nostro caso si utilizza la funzione *thetaf()* disponibile nel pacchetto R *forecast* (R. J. Hyndman e Khandakar 2008).

1.3 Metodo Naive stagionale

È un metodo particolarmente utile per serie storiche contenenti una forte componente stagionale. In questo caso, si fissa ogni previsione futura uguale all'ultimo valore osservato nello stesso periodo. Formalmente, la previsione al tempo $T + h$ per la serie storica $\{y_i\}_{i=1}^{T}$ risulta:

$$\hat{y}_{T+h|T} = y_{T+h-s(k+1)}$$

dove s è il periodo stagionale e k è la parte intera di $(h-1)/s$ (per esempio il numero di anni completamente trascorsi nel periodo di previsione prima di $T + h$). Il modello assume la forma di un $ARIMA(0,0,0)(0,1,0)_s$. Il tutto implementato attraverso la funzione *snaive* disponibile nel pacchetto *forecast* di R (R. J. Hyndman e Khandakar 2008).

1.4 Modello ETS

I modelli di lisciamento esponenziale, considerando l'intero orizzonte di dati storici disponibili, assegnano un

peso decrescente in modo esponenziale alle osservazioni della serie storica, ipotizzando che i valori più recenti assumano maggior importanza rispetto a valori più vecchi nella previsione del prossimo valore futuro. In altre parole, si tratta di una tecnica che consiste nell'effettuare una media ponderata di tutte le osservazioni disponibili, con peso variabile in base alla prossimità temporale. Data la natura non parametrica del modello sono escluse tutte le considerazioni probabilistiche usuali. In questo contesto si ricorre alla specificazione *state-space* dei modelli, per cui una serie storica $\{y_t\}_{t=1}^{T}$ è associata ad una serie di vettori non osservabili, che descrive lo stato della variabile in un determinato istante temporale in funzione dei precedenti stati e di un termine di errore. Tale relazione viene specificata tramite una equazione di previsione, mentre l'equazione di livellamento descrive l'evoluzione del sistema dinamico nel tempo. Nei modelli *state-space* gli errori possono essere di natura additiva (A) o moltiplicativa (M), tale scelta comporta i medesimi valori per la previsione puntuale ma intervalli di previsione differenti. Nell'esperimento che conduciamo la scelta sarà automatica. I modelli di lisciamento esponenziale permettono di specificare la relazione tra le componenti latenti che lo compongono per cui oltre agli errori e alla loro natura, è possibile definire diversi metodi prendendo in considerazioni differenti forme della componente di trend e/o della stagionalità. La funzione *ets* del pacchetto *forecast* (Hyndman e Khandakar 2008) di R imposta di default un modello *ETS(Z,Z,Z)*, per cui la scelta della natura rispettivamente di errore, trend e stagionalità è automatica, se non viene rilevata la componente si associa la lettera N.

Il modello più comunemente utilizzato è il modello di

Brown e G. (1957), chiamato anche Lisciamento Esponenziale Semplice (ETS(A,N,N)). La previsione per il periodo $t+1$ è descritta dall'equazione di livellamento ricorsiva:

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}$$

per $t = 1, ..., T$ dove α è un coefficiente di ponderazione e lisciamento dei due termini relativi al tempo t. Si noti che al crescere del peso α, la rilevanza attribuita dal modello all'ultimo valore osservato aumenta. Ciò significa che la previsione è reattiva nei confronti di variazioni improvvise del fenomeno osservato. Sviluppando l'equazione ricorsiva, si può scrivere sinteticamente:

$$\hat{y}_{t+1} = \alpha \sum_{h=0}^{T-1} (1-\alpha)^h y_{t-h}$$

con $h \geq 1$.

Il lisciamento esponenziale semplice fornisce delle previsioni costanti, ragion per cui è sufficiente calcolare la previsione della serie storica in esame un solo passo avanti. Il valore ottimale di α è quel valore che minimizza una funzione obbiettivo (come l'errore quadratico medio):

$$\hat{\alpha} = \arg\min_{\alpha} \left\{ \sum_{t=m}^{T-1} (y_{t+1} - \hat{y}_{t+1})^2 \right\}$$

dove m rappresenta il numero di termini da trascurare per evitare gli effetti di un valore iniziale predeterminato. Di default nella funzione *ets()* viene scelto il valore che massimizza la log-verosimiglianza. In questa situazione potremmo specificare come equazione di previsione $\hat{y}_{t+h|t} = l_t$ ed equazione di livellamento $l_t = \alpha y_t + (1-\alpha) l_{t-1}$.

Holt (1957) ha esteso il lisciamento esponenziale semplice per consentire la previsione dei dati in presenza del-

la componente di trend. Con questo metodo le equazioni di livellamento sono due:

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$$
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

e l'equazione di previsione diventa:

$$\hat{y}_{t+h|t} = l_t + hb_t$$

In questo caso β è il parametro di lisciamento per il trend.

Le previsioni generate dal metodo di Holt mostrano una tendenza costante (in aumento o in diminuzione) per un periodo indefinito nel futuro. L'evidenza empirica indica che questi metodi tendono a sovrastimare, soprattutto per orizzonti previsivi lunghi. Motivati da questa osservazione, Gardner Jr e McKenzie (1985) hanno introdotto un parametro che 'smorza' la tendenza ad una linea piatta in un momento futuro. In aggiunta ad α e β inseriamo un parametro $\phi \in [0;1]$:

$$\hat{y}_{t+h|t} = l_t + (\phi + \phi^2 + ... + \phi^h)b_t$$
$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1})$$
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)\phi b_{t-1}$$

La previsione converge a $l_T + \phi b_T/(1-\phi)$ quando $h \to \infty$ confermando che le previsioni nel breve-medio termine sono caratterizzate da trend mentre nel lungo termine sono costanti.

Holt e Winters (1960) hanno esteso il metodo di Holt per catturare la stagionalità (s_t con frequenza m). È bene sottolineare che si preferisce il metodo additivo quando le variazioni stagionali sono costanti attraverso la serie, mentre il modello moltiplicativo è preferito quando le variazioni stagionali cambiano proporzionalmente al livello della serie. Le equazioni si aggiornano come segue:

$$
\begin{aligned}
\hat{y}_{t+h|t} &= l_t + hb_t + s_{t+h-m(k+1)} \\
l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\
b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\
s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}
\end{aligned}
$$

dove γ livella la stagionalità e k è la parte intera di $(h-1)/m$.

In generale sono possibili diverse combinazioni delle tre componenti anche tenendo conto della loro natura e dell'inserimento o meno di una costante.

1.5 Modello TBATS

Il metodo *TBATS* (Hyndman et al. 2011) rappresenta un'innovazione dei modelli *state-space* in grado di gestire situazioni di multi-stagionalità e alcuni tipi di non linearità utilizzando il metodo di Box-Cox (Box e Cox 1964). Il modello *TBATS* scompone una serie temporale in quattro componenti: trend, stagionalità, errore e una componente aggiuntiva per catturare qualsiasi variazione residua nei dati. La componente di trend può essere modellata come una funzione lineare o non lineare, mentre la componente di stagionalità può catturare modelli stagionali multipli con frequenze, ampiezze e fasi diverse utilizzando funzioni trigonometriche. La componente di errore è modellata come un processo ARMA e la componente aggiuntiva è modellata utilizzando una trasformazione Box-Cox per gestire l'eteroschedasticità. Sia y_t l'osservazione al tempo t della serie osservata. Indichiamo con $y_t^{(\omega)}$ il valore di y_t a seguito della trasformazione di Box-Cox con parametro ω. Il modello include quin-

di una trasformazione di Box-Cox, un modello ARMA per gli errori e modelli stagionali le cui componenti sono basate sulla serie di Fourier come segue:

$$
\begin{aligned}
y_t^{(\omega)} &= \begin{cases} \frac{y_t^{\omega}-1}{\omega} & \omega \neq 0 \\ \log(\mathrm{y_t}) & \omega = 0 \end{cases} \\
y_t^{(\omega)} &= l_{t-1} + \phi b_{t-1} + \sum_{i=1}^{T} s_{t-1}^{(i)} + d_t \\
l_t &= l_{t-1} + \phi b_{t-1} + \beta d_t \\
b_t &= (1-\phi)b + \phi b_{t-1} + \beta d_t \\
s_t^{(i)} &= \sum_{j=1}^{k_i} s_{j,t}^{(i)} \\
s_{j,t}^{(i)} &= s_{j,t-1}^{(i)} \cos(\lambda_j^{(i)}) + s_{j,t-1}^{*(i)} \sin(\lambda_j^{(i)}) + \gamma_1^{(}i) d_t \\
s_{j,t}^{*(i)} &= -s_{j,t-1} \sin(\lambda_j^{(i)}) + s_{j,t-1}^{*(i)} \cos(\lambda_j^{(i)}) + \gamma_2^{(i)} d_t \\
d_t &= \sum_{i=1}^{p} \varphi_i d_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t
\end{aligned}
$$

Dove l_t è la componente di livello al tempo t, b è il trend di lungo periodo, b_t è il trend nel breve termine nel periodo t, $\lambda_j^{(i)} = 2\pi j/m_i$ dove $m_1, ..., m_T$ denotano i diversi periodi stagionali, d_t è un processo $ARMA(p,q)$ e $\epsilon_t \sim WN(0, \sigma^2)$. I parametri di lisciamento sono α, β, γ_1 e γ_2. Inoltre il modello si serve del parametro ϕ di *Gardner e McKenzie* per smorzare il trend all'aumentare dell'orizzonte di previsione, ma integrato con un trend di lungo periodo b. Per quanto riguarda la componente stagionale, descriviamo il valore della i-esima componente stagionale con $s_{j,t}^{(i)}$ e il cambiamento del suo livello nel tempo con $s_{j,t}^{*(i)}$. Il numero di serie armoniche richieste dall'i-esima componente stagionale è k_i. Si ottiene così il modello $TBATS(\omega, \phi, p, q, m_1, k_1, ..., m_T, k_T)$ che sta

per *Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components.*

Si utilizza una combinazione di funzioni trigonometriche per catturare i modelli stagionali che si verificano su diverse frequenze (ad esempio, settimanali, mensili e annuali). Si applica una trasformazione Box-Cox per stabilizzare la varianza delle serie temporali. La componente ARMA degli errori del modello tiene conto dell'autocorrelazione residua nelle serie temporali. Oltre alle componenti stagionali e di errore, il metodo *TBATS* include componenti di tendenza, che possono catturare tendenze lineari o non lineari nei dati. Il modello stima i parametri di queste componenti col metodo della massima verosimiglianza. Viene implementato con la funzione *tbats* disponibile nel pacchetto *forecast* (R. J. Hyndman e Khandakar 2008) di R.

1.6 Rule-Based Tree

I *Rule-Based Tree* sono un metodo supervisionato rappresentante un'estensione del modello ad albero M5 di Quinlan (Quinlan et al. 1992), implementato nel pacchetto R *Cubist* (Kuhn et al. 2014) .

L'idea è quella di costruire un albero avente su ciascun nodo e foglia un modello di regressione impiegato per fare la previsione e che infine quest'ultima venga lisciata.

Supponiamo di avere un insieme di addestramento T, per coltivare l'albero si parte dalla radice contenente tutte le variabili in T e si divide in due sottogruppi che apparterranno ai due nodi figli. Siano T_i il sottogruppo di casi che hanno l'i-esimo risultato del test e $sd(T_i)$ la deviazione standard della variabile dipendente in T_i. Per

effettuare la divisione si cerca tra tutte le possibili variabili, il valore della variabile tale che sia massimizzata la riduzione dell'errore previsto:

$$\Delta error = sd(T) - \sum_i \frac{T_i}{T} sd(T_i)$$

Si procede fino ad un criterio di arresto se specificato, altrimenti fino ad ottenere l'albero avente sulle foglie tutti i singoli valori della variabile risposta. Per ogni nodo dell'albero vengono stimati modelli di regressione lineare utilizzando solo le variabili che sono giunte presso quello specifico nodo, successivamente ogni modello viene semplificato con una procedura *backward*. L'albero viene poi potato per evitare problemi di sovradattamento. L'ultima fase di *smoothing* serve per compensare le discontinuità che inevitabilmente si andranno a creare tra modelli adiacenti ai nodi dell'albero potato. Si utilizza prima il modello della foglia per calcolare il valore previsto e poi si filtra tale valore lungo il percorso a ritroso verso la radice, combinandolo con il valore previsto dal modello lineare per quel nodo che intercetta. La formula di lisciamento è:

$$p^* = \frac{np + kq}{n + k}$$

dove p^* è il predittore passato al nodo successivo, p è il predittore proveniente dal nodo precedente, q è il predittore del nodo di riferimento, n è il numero di istanze che raggiungono il nodo e k una costante con valore di default pari a 15.

L'albero viene fatto collassare in un insieme di regole, dove una regola è un'affermazione logica condizionale che mappa un insieme di condizioni sulle variabili di ingresso a una decisione specifica. Ogni nodo è una regola della forma *if-then* che delinea una ramificazione, rispetto agli

alberi o ad altri modelli quindi è più facile spiegare il processo a chiunque.

Un'implementazione caratteristica dei *Cubist Base-Model* riguarda i *committees* (comitati). Questa opzione permette di creare iterativamente modelli ad albero in sequenza. Il primo albero generato segue la procedura descritta in precedenza, gli alberi successivi invece vengono creati utilizzando versioni adattate al risultato del set di dati di addestramento, ad esempio, se un campione è stato sottostimato in precedenza, il suo risultato viene modificato affinché sia più grande e il modello venga tirato verso l'alto nel tentativo di interrompere la sottostima. Nei comitati il primo modello usa il valore del risultato originale y_i, nelle iterazioni successive viene utilizzato al suo posto un valore modificato y_i^*. Per l'mesima iterazione il modello imposta la seguente formula di aggiustamento: $y^*_{i,(m)} = y_i - (\hat{y}_{i,(m-1)} - y_i)$.

Un maggior numero di *committees* porta in genere a una miglior accuratezza, ma al costo di un aumento del tempo di addestramento e della complessità.

1.7 Random Forest

Il termine *Random Forest* (foreste casuali) si riferisce ad un metodo statistico non parametrico che affronta i problemi di regressione e classificazione, in grado di gestire relazioni sia lineari che non, tra le variabili input e quelle obbiettivo.

Introduciamo il concetto di *bagging* (o *boostrap aggregation*) ossia una tecnica per ridurre la varianza di una funzione di previsione stimata: per la regressione, si trat-

ta di applicare molte volte lo stesso albero di regressione su versioni campionate via *bootsrap* dell'insieme dei dati di addestramento e si calcola la media dei risultati. Quindi l'idea è quella di calcolare la media di molti modelli contenenti errore, ma approssimativamente non distorti, in modo da ridurre la varianza.

Le foreste casuali sono una variante del metodo di *bagging* che coltiva una grande raccolta di alberi (anche detta foresta) e ne calcola la media. Poiché ognuno dei B alberi generati da questa procedura proviene da una distribuzione identica, il valore atteso della media di tutti gli alberi sarà pari al valore atteso di ciascuno. L'unica opzione di miglioramento avviene tramite la riduzione della varianza, infatti sia σ^2 la varianza di ogni albero della foresta i.i.d., allora la media delle B previsioni di ciascun albero avrà varianza pari a $\frac{1}{B}\sigma^2$; se le previsioni sono solo i.d. con correlazione $\rho > 0$, la varianza della media è $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. Si intuisce che all'aumentare di B il secondo termine della somma decresce.

L'obbiettivo delle foreste casuali è quello di migliorare la riduzione della varianza del *begging* riducendo la correlazione tra gli alberi, senza incrementare troppo la varianza. Per fare ciò si selezionano le variabili casualmente per poi far crescere i B alberi. La previsione di una foresta casuale è:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b)$$

dove $T(x;\Theta_b)$ rappresenta l'albero cresciuto sul Θ_b insieme di variabili. L'algoritmo del *Random Forest* può essere riassunto nei seguenti passi:

1. per $b = 1, ..., B$:

(a) estrai un campione *bootstrap* di dimensione N dai dati di addestramento.

(b) fai crescere un albero T_b della foresta casuale su dati surrogati, ripetendo ricorsivamente i seguenti passaggi per ciascun nodo terminale dell'albero fino alla dimensione minima dei nodi n_{min}:

i. seleziona m variabili a caso tra le p variabili.

ii. seleziona la miglior variabile e punto di divisione tra le m.

iii. dividi il nodo in due nodi figli.

2. ritorna l'insieme degli alberi $\{T_b\}_1^B$.

Per prevedere il valore della variabile risposta si fa percorrere alle covariate ognuno degli alberi appartenenti all'insieme finale ottenendo B stime, poi se ne effettua la media che corrisponderà alla previsione finale. L'algoritmo è implementato nel pacchetto R *ranger* (Wright e Ziegler 2015) con la omonima funzione.

1.8 Multivariate Adaptive Regression Splines

La *Multivariate Adaptive Regression Splines* (MARS) è una tecnica di regressione non parametrica introdotta da Friedman 1991 e può essere vista come un'estensione della regressione spline.

La spline è una funzione definita a tratti da polinomi utile per studiare la relazione tra due variabili X e Y per le quali osserviamo le coppie (x_i, y_i) per $i = 1, ..., n$. In particolare è di interesse studiare E(Y|X = x) quando l'assunzione di linearità non è soddisfatta. Un modello appropriato per i dati è:

$$y = f(x; \beta) + \varepsilon$$

con $\varepsilon \sim$ IID $N(0, \sigma^2)$. L'idea è di sezionare l'asse x in $K+1$ intervalli separati da K nodi ξ ed interpolare gli n punti. Per la stima del comportamento strutturale di Y viene scelta una combinazione lineare di funzioni semplici, ma non correlate (problema di multicollinearità). Col fine di regolare il grado di lisciatura della stima $\hat{f}(x)$ viene minimizzata la funzione obbiettivo:

$$\sum_{i=1}^{n}\{y_i - f(x)\}^2 + \lambda \int f''(x)^2 dx$$

dove $\int$f"(x)^{2}dx è tanto più grande quanto meno $f(\cdot)$ è liscia. Il parametro λ decide quanto penalizzare la funzione per la sua "non lisciatura". Se $\lambda \to +\infty$, si riottiene la regressione lineare semplice. Al contrario, se $\lambda \to 0$ si ottiene un adattamento interpolatorio che cambia direzione di movimento senza alcun vincolo. Green e Silverman (Green e Silverman 1993) mostrano come la soluzione al problema di minimizzazione è rappresentato da una spline cubica ottenendo un modello del tipo

$$f(x) = \sum_{j=0}^{n_0} \beta_j B_j(x)$$

dove le funzioni base $B_j(\cdot)$ vengono combinate con pesi decisi dai parametri β_j e n_0 rappresenta il numero distinto di punti x_i. Possiamo riscriverla come:

$$D(f, \lambda) = (y - B\beta)^T (y - B\beta) + \lambda \beta^T \Omega \beta$$

dove B è la matrice contenente sulle colonne i valori $B_j(x)$ dei rispettivi valori di x, e Ω è la matrice contenente il generico elemento $\int B_j$"(x)B_k"(x)dx. La soluzione del problema di ottimizzazione produce

$$\hat{\beta} = (B^T B + \lambda \Omega)^{-1} B^T y$$

Tuttavia dal punto di vista computazionale risulta costoso trattare una matrice di ordine n_0, inoltre estendendo le *splines* a casi con due o più covariate la funzione di penalizzazione si complica maggiormente. In letteratura esistono algoritmi più efficienti che permettono di trattare questo problema, in particolare quando il numero di covariate è elevato si può ricorrere al *Multivariate Adaptive Regression Splines*. L'obbiettivo è di trovare una relazione tra la variabile dipendente y e le p covariate $x = (x_1, ..., x_p)^T$. Per ogni variabile esplicativa x_j determiniamo una coppia di funzioni base del tipo $(x_j - \xi)_+$ e $(\xi - x_j)_+$ dove $(x_j - \xi)_+ = \max\{x_j - \xi, 0\}$ e $(\xi - x_j)_+ = \max\{\xi - x_j, 0\}$ con il nodo $\xi \in \{x_{1j}, x_{2j}, ..., x_{nj}\}$ per $j = 1, .., p$. Otteniamo l'insieme:

$$C = \{x_j, (x_j - \xi)_+, (\xi - x_j)_+ : \xi \in (x_{i1}, ..., x_{ip}), i = 1, ..., n, j = 1, ..., p\}$$

contenente $2np$ funzione base. Bisogna selezionare un sottoinsieme di C da combinare in un modello che si adatti ai dati. Il modello *MARS* ha una forma del tipo

$$f(x) = \beta_0 + \sum_{k=1}^{2K} \beta_k h_k(x)$$

dove $h_k(x)$ è una funzione in C o il prodotto di più funzioni. Per stimare $h_k(x)$ e i parametri β_k si esegue il seguente processo ricorsivo:

- Si pone $K = 0$ e si introduce la funzione costante $h_0(x) = 1$.
- Incrementiamo il valore di K e supponiamo di aver introdotto nel modello $2(K-1)$ termini. Scegliamo di aggiungere nel modello la coppia di funzioni che minimizza il criterio dei minimi quadrati.

- Il processo continua fino ad un prefissato K

Al termine del processo avremmo un modello che tipicamente produce un sovradattamento dei dati, quindi adottiamo una procedura di eliminazione *backward* o *forward* ottenendo il modello migliore $\hat{f}_\lambda(x)$ per ogni λ.
λ rappresenta il costo per grado di libertà imposto, per la sua stima si potrebbe usare la convalida incrociata ma, per motivi computazionali, il *MARS* utilizza una sua generalizzazione. Viene selezionato lungo il percorso di selezione *backward*, il modello che minimizza:

$$GCV(\lambda) = \frac{\sum_{i=1}^{n}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/n)^2}$$

$M(\lambda)$ è il numero effettivo di parametri nel modello. Una proprietà chiave del *MARS* è la sua abilità nell'operare localmente, le funzioni che lo compongono valgono zero al di fuori del loro raggio d'azione, il tutto con una complessità pari a $O(n)$ operazioni.

Per l'adattamento del modello si usa la funzione *earth()* appartenente all'omonimo pacchetto R (Milborrow et al. 2017) .

1.9 Processi Gaussiani

I processi Gaussiani rappresentano una classe di modelli di funzioni che è allo stesso tempo semplice e generale. In termini più precisi, un processo Gaussiano può essere definito come una distribuzione di funzioni in cui ogni insieme finito di valori di funzioni, come f(x1), ..., f(xn), segue una distribuzione congiunta gaussiana. Un processo gaussiano è completamente specificato dalla sua funzione media e dalla sua funzione di covarianza, anche nota come *kernel*.

La funzione media di un processo Gaussiano, indicata come $E(f(x))$, assegna un valore medio a ciascun punto nello spazio. Di solito, si assume una media di valore zero in tutti i punti, poiché la sua incertezza può essere tenuta in considerazione aggiungendo un termine extra al kernel. D'altra parte, la funzione di covarianza, $Cov(f(x), f(x^*))$, specifica come i punti nel processo sono correlati tra loro. La scelta del kernel determina la struttura che il modello di processo Gaussiano può catturare. Esistono diverse funzioni di covarianza tra cui scegliere, e il kernel appropriato può essere selezionato per specificare una vasta gamma di modelli.

Un processo Gaussiano descrive una distribuzione di probabilità sulle possibili funzioni che si adattano a un insieme di punti. Si basa sull'assunzione che le osservazioni adiacenti dovrebbero trasmettere informazioni l'una all'altra. In particolare, si assume che le variabili osservate siano normalmente distribuite e che l'accoppiamento tra di esse avvenga attraverso la matrice di covarianza.

La funzione di regressione modellata da un processo Gaussiano può essere espressa come

$$P(f|X) = N(f|\mu, K)$$

Per cui $X = [x_1, ..., x_n]$ sono i dati osservati, $f = [f(x_1), ..., f(x_n)]$, $\mu = [m(x_1), ..., m(x_n)]$, e $K_{ij} = k(x_i, x_j)$, dove m rappresenta la funzione media e k indica la funzione di kernel. Il processo Gaussiano è una distribuzione sulle funzioni, in cui la sua forma è influenzata dal kernel K. Se due punti x_i e x_j sono considerati "simili" dalla funzione di kernel, i rispettivi output della funzione $f(x_i)$ e $f(x_j)$ dovrebbero essere simili.

Nell'applicazione dei processi Gaussiani, sono state considerate quattro funzioni kernel per la stima e la pre-

visione. Una delle più comuni è la "radial basis function" (RBF), che è espressa come $k(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right)$. Altre funzioni kernel utilizzate sono la "vanilladot" (kernel lineare), la "polydot" (kernel polinomiale) e la "laplacedot" (kernel laplaciano).

Un aspetto importante dei processi Gaussiani è che a differenza delle gaussiane multivariate che catturano un numero finito di gaussiane distribuite congiuntamente, il processo Gaussiano non ha questa limitazione. La sua media e covarianza sono definite da una funzione che può avere un dominio infinito. Di conseguenza, il processo Gaussiano può essere interpretato come una variabile casuale gaussiana di dimensione infinita.

Un vantaggio dei processi Gaussiani è la loro flessibilità nel modellare una vasta gamma di fenomeni. Trattando il processo Gaussiano come definito a priori dalla funzione di kernel, possiamo ottenere una distribuzione a posteriori una volta disponibili i dati. Questa distribuzione a posteriori può essere utilizzata per prevedere il valore atteso e la probabilità della variabile di output, dato un certo insieme di variabili di input.
Per comprendere meglio come un processo Gaussiano modella una funzione unidimensionale, consideriamo il seguente esempio:

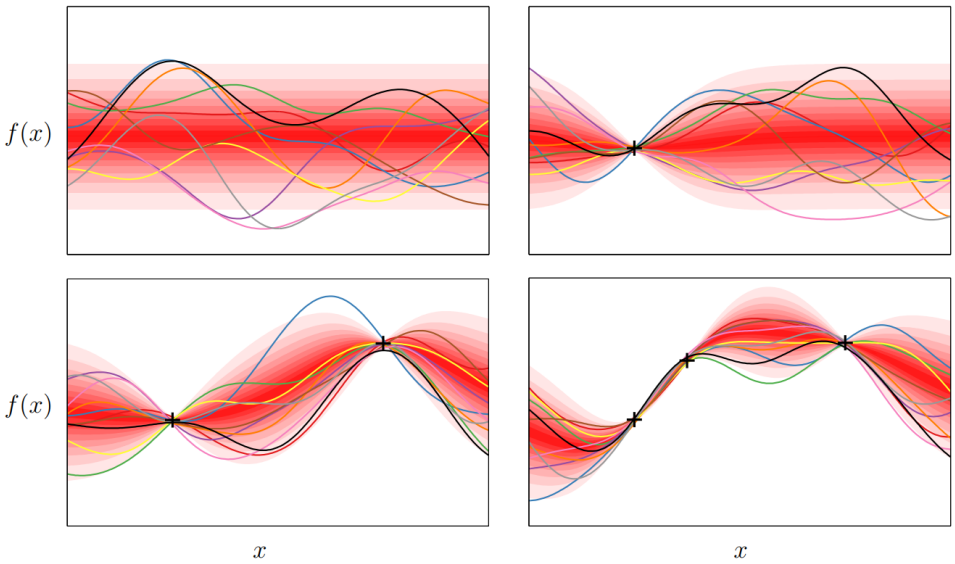

Figura 1: Una rappresentazione visiva di un processo Gaussiano che modella una funzione unidimensionale

Nell'immagine, le diverse intensità di rosso sono associate ai decili della funzione di densità in ogni punto dell'insieme di input. Le diverse linee colorate rappresentano alcune delle ipotesi incluse nel modello. Le immagini mostrano l'evoluzione del modello dopo il condizionamento con nuovi dati, osservando come si adatta alla struttura dei dati.
In conclusione, un processo Gaussiano è una distribuzione di funzioni completamente specificata da una funzione media e da una funzione di covarianza. Le previsioni effettuate da un processo Gaussiano sono medie ponderate dei dati osservati, in cui il peso è determinato dalla funzione media e dalla funzione di covarianza. Per l'implementazione si impiega la funzione *gausspr()* disponibile nel pacchetto R *kernlab* (Karatzoglou et al. 2004).

1.10 Modelli Lineari Generalizzati

I modelli in questione si occupano di trovare con la regolarizzazione un compromesso tra varianza e distorsione nei problemi di regressione lineare. Regolarizzare uno stimatore, significa introdurre un termine di regolarizzazione (o penalizzazione) nella funzione utilizzata per addestrare il modello. Regolarizzando (comprimendo, controllando, ecc...) in qualche modo lo stimatore, la sua varianza sarà ridotta.

Supponiamo sia Y la variabile risposta e di voler prevedere il suo valore con una combinazione lineare di p covariate. Il modello è del tipo $Y = X\beta + \epsilon$ con $\epsilon \sim N(0, \sigma^2)$. Non conoscendo i veri valori dei parametri β, il metodo dei minimi quadrati li stima minimizzando la funzione obbiettivo $||y - X\hat{\beta}||^2$ ottenendo la stima dei parametri come $\hat{\beta} = (X^T X)^{-1} X^T Y$.

Assumiamo ora di conoscere il vero valore dello stimatore che vale β_0. Una buona misura della qualità di uno stimatore è l'errore quadratico medio:

$$MSE\left\{\hat{\beta}\right\} = E\left\{[\hat{\beta} - \beta_0]^2\right\} = Var\left\{\hat{\beta}\right\} + E\left[\hat{\beta} - \beta_0\right]^2 \tag{1.1}$$

Dal *Teorema di Gauss-Markov* la previsione con il metodo dei minimi quadrati porta ad ottenere lo stimatore non distorto e con varianza minore tra tutti gli stimatori che siano funzioni lineari di y. Tuttavia esistono stimatori distorti con MSE più piccolo che si ottengono regolarizzando lo stimatore: si riduce la varianza se il corrispondente incremento in distorsione è piccolo. I metodi utilizzati nell'esperimento sono quelli della *regressione ridge*, *regressione lasso* e l'*elastic net*.

La regressione ridge risolve il problema di minimo

$$\min_{\beta}(y - X\beta)^T(y - X\beta)$$

soggetta al vincolo $\sum_{j=1}^{p} \beta_j^2 \leq \lambda$ con $\lambda > 0$ scelto in modo tale da bilanciare varianza e distorsione. La soluzione è lineare ed esplicita: $\hat{\beta}_\lambda = (X^T X + \lambda I)^{-1} X^T y$ tale per cui se $\lambda = 0$ si ottiene lo stimatore ai minimi quadrati. Le formula della varianza per i coefficienti vale:

$$Var(\hat{\beta}_{ridge}) = \sigma^2 (X^T X + \lambda I)^{-1} X^T X (X^T X + \lambda I)^{-1}$$

È facile intuire che all'aumentare di λ la varianza decresce. Per la scelta di λ un approccio *machine learning* suggerisce di scegliere il valore tale per cui viene minimizzata la somma dei quadrati dei residui ottenuta dalla convalida incrociata, altrimenti un approccio più tradizionale sceglie il valore che minimizza il criterio di informazione di Akaike (AIC). La regressione ridge comprime le componenti con bassa varianza più di quelle con varianza maggiore quindi cerca di comprimere a zero gli stimatori riducendo drasticamente la varianza.

La regressione lasso è concettualmente simile alla ridge ma in questo caso se λ è sufficientemente piccolo, alcuni coefficienti vengono 'stimati' esattamente a zero. Si occupa di risolvere il problema di minimo:

$$\min_{\beta}(y - X\beta)^T(y - X\beta)$$

soggetto al vincolo $\sum_{j=1}^{p} |\beta_j| \leq \lambda$. La soluzione non è lineare in y ed è necessario ricorrere ad algoritmi di programmazione quadratica.

Alcuni aspetti di confronto da considerare sono che la lasso può impostare alcuni coefficienti a zero, lavora bene

quando ci sono pochi parametri significativi e i restanti sono vicini allo zero, mentre la ridge non può impostare i coefficienti a zero e lavora meglio quando ci sono molti parametri grandi significativi che impattano sulla risposta.

Una combinazione di ridge e lasso emerge combinando le penalizzazioni dei due modelli nelle *Elastic Net*. La funzione obbiettivo da minimizzare è la seguente:

$$\frac{\sum_{i=1}^{n}(y_i - x_i^T\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{p}\hat{\beta}_j^{\,2} + \alpha\sum_{j=1}^{p}|\hat{\beta}_j|\right)$$

Dove α è il parametro che assegna i pesi a ridge e lasso. La funzione *glmnet* (J. Friedman, Hastie e Tibshirani 2010) presente nell'omonimo pacchetto R utilizzato per adattare il modello, il parametro λ viene stimato con la convalida incrociata mentre α è fissato a priori, in particolare si testano modelli con $\alpha = 0, 0.25, 0.5, 0.75, 1$.

Capitolo 2

Metodologia

In letteratura sono stati proposti diversi metodi per l'analisi delle serie storiche con l'obbiettivo di fare previsione o di aiutare a comprendere la struttura sottostante dei dati. Sia $Y = y_1, ..., y_n$ una serie storica, la procedura di previsione mira a stimare i valori futuri di Y, y_{n+h} dove h denota l'orizzonte previsivo. Esistono serie storiche univariate e multivariate: le prime sfruttano per la stima dei valori futuri la struttura individuata dai dati osservati, le seconde invece estendono questo approccio prendendo in considerazione ulteriori serie temporali aggiuntive che vengono utilizzate come variabili esplicative. Una considerazione fondamentale riguarda l'orizzonte di previsione: più esso aumenta, cioè più cerchiamo di prevedere valori lontani nel tempo, maggiore sarà la difficoltà e l'incertezza nell'effettuare le previsioni. A tal proposito è di interesse valutare la capacità predittiva dei modelli sia sull'osservazione futura più prossima, cioè un passo in avanti, sia sui valori futuri distanti, motivo per cui i modelli verranno utilizzati per prevedere i valori delle successive 18 osservazioni rispetto all'ultima unità che è entrata a far parte dell'insieme di stima.

2.1 Dati

Per la replicazione dell'esperimento vengono considerate solo serie storiche univariate senza dati mancanti provenienti dal pacchetto *Time Series Data Library* (tsdl, R. Hyndman 2014). Si prendono tutte le serie storiche univariate con almeno 1000 osservazioni. Esse possiedono frequenze di campionamento differenti (giornaliera, mensile, ecc) e provengono da diversi ambiti di applicazione (finanza, fisica, ecc), è possibile trovare una descrizione completa presso la relativa fonte del database. Alle serie storiche considerate ne sono state aggiunte 35 utilizzate in 'Cerqueira, Torgo e Soares (2019)' che non fossero già presenti. Alla fine l'analisi si baserà su 90 serie temporali troncate a 1000 osservazioni.

Invece la seconda parte, si basa su serie storiche troncate alla 500-esima osservazione, non considerate dagli autori e provenienti da differenti pacchetti R: *TSA* (Chan et al. 2022), *forecast* (R. J. Hyndman e Khandakar 2008), *fpp2* (R. Hyndman, Athanasopoulos, GGally et al. 2022), *ggplot2* (Wickham 2011), *datasets* (Antal 2022), *astsa* (D. Stoffer e M. Stoffer 2023) e *tsdl* (R. Hyndman 2014). Per un totale di 80 serie storiche provenienti anch'esse da svariati ambiti di applicazione e con frequenze di campionamento differenti.

2.2 Pre-elaborazione

Ogni serie storica del campione, col fine di produrre un miglior adattamento del modello, viene pre-elaborata. In particolare per stabilizzare la varianza si utilizza la trasformazione di Box-Cox il cui parametro di trasforma-

zione viene ottimizzato secondo *Guerrero (1993)*; questo metodo cerca di minimizzare il coefficiente di variazione ossia una misura standardizzata di dispersione definita come il rapporto tra la deviazione standard e la media. In secondo luogo si tratta la stagionalità prima rilevandola col *test di X. Wang, Smith e R. Hyndman (2006)* e qualora fosse presente, decomponendo la serie col metodo moltiplicativo. Questo processo viene saltato per i modelli *SARIMA* ed *ETS* in quanto dispongono già di metodi per la rimozione della stagionalità. Infine si applica il *test di Cox-Stuart* per capire se rimuovere la componente di trend con la differenza prima. Questa procedura è stata applicata ad entrambi i metodi.

2.3 Metodo di valutazione

Per quanto riguarda la metodologia di stima assunta si usa una procedura prequenziale. Una procedura prequenziale è un metodo di valutazione in cui un'osservazione viene prima utilizzata per testare un modello predittivo e successivamente diventa parte del set di training e viene usata per aggiornare il modello. Può essere utilizzata per costruire una curva di apprendimento, ovvero un insieme di punteggi di performance di un modello predittivo, in cui l'insieme è ordinato man mano che la dimensione del campione cresce. In altre parole, la procedura prequenziale consente di valutare come la performance di un modello predittivo cambia all'aumentare dell'insieme di stima. Nel nostro caso vengono utilizzate le prime n-18 osservazioni per stimare il modello e successivamente si applica la procedura prequenziale descritta. Come metrica di valutazione ho usato il *Mean Absolute Scaled*

Error (MASE) cioè l'errore assoluto medio dei valori di previsione, diviso per l'errore assoluto medio della previsione *naive* a un passo nel campione; però principalmente il confronto è avvenuto utilizzando il metodo del rango. Il rango (*rank* in inglese) è un metodo non parametrico che consiste nel classificare le prestazioni di ciascun approccio previsionale in base a un determinato criterio. Si definisce come metrica di valutazione il *MASE* che viene calcolato per ogni modello ad ogni passo della procedura di previsione fino alla stima dell'ultima osservazione. Poi si classificano gli approcci in base alla loro performance sulla metrica scelta affinché quello con le migliori prestazioni riceva la posizione più alta, mentre l'approccio con le peggiori prestazioni abbia la posizione più bassa. Se un modello supera costantemente gli altri, avrà un rango più alto (cioè pari a 1), viceversa se è costantemente peggiore avrà un rango basso (pari a 10). Se i ranghi sono simili, è probabile che gli approcci previsionali siano comparabili in termini di accuratezza. Ad ogni passo viene calcolata la media, per ogni modello, del suo rango sulle 90 (o 80) serie storiche ottenendo il rango medio (*average rank*). Per rendere più semplice l'interpretazione grafica, il rango medio viene lisciato con una media mobile di ordine 50.

Nel caso delle previsione a più passi in avanti il discorso si complica. Si misura l'accuratezza di ciascuna strategia di previsione utilizzando il *MASE*. Per verificare se esistono differenze significative tra le performance dei modelli, bisogna risolvere il problema del confronto tra più modelli che effettuato previsione multiple su più serie di dati. Tale problema é risolto con un procedimento in due fasi: prima si applica il *test di Friedman* per valutare se i modelli hanno lo stesso rango medio; se

il test rifiuta l'ipotesi nulla è necessario eseguire il *test post-hoc a coppie* per confrontare i diversi modelli.

2.3.1 Test di Friedman

Il test di Friedman è una procedura non parametrica che verifica la significatività delle differenze tra più classifiche. Classifica gli algoritmi per ogni set di dati separatamente: il grado 1 viene assegnato all'algoritmo con le migliori prestazioni, il grado 2 al secondo migliore e così via. Si noti che in caso di parità viene assegnata una media dei ranghi. Dopo aver classificato gli algoritmi per ogni set di dati, il test confronta i ranghi medi degli algoritmi. Sia r_j^i il rango del j-esimo modello (di k totali) sull'i-esimo insieme di verifica, il rango medio del j-esimo metodo è $R_j = \frac{1}{N}\sum_i r_j^i$. L'ipotesi nulla prevede che tutti gli algoritmi siano equivalenti perciò il loro rango medio dovrebbe essere uguale. Sotto l'ipotesi nulla la statistica di Friedman è:

$$Q = \frac{12N}{k(k+1)}\sum_j R_j^2 - \frac{k(k+1)^2}{4}$$

che si distribuisce come una Chi-quadro con $k-1$ gradi di libertà.

2.3.2 Test post-hoc a coppie

Quando l'ipotesi nulla del test di Friedman viene rifiutata, si applica il test post-hoc a coppie per identificare le differenze significative tra tutti gli algoritmi. La statistica test per confrontare l'i-esimo metodo col j-esimo

è:

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N}}}$$

distribuita asintoticamente come una normale.

Capitolo 3

Risultati delle analisi empiriche

Il capitolo riporta i risultati ottenuti da Cerqueira et al. (2019), si occupa di fornire una panoramica sull'esperimento effettuato dagli autori, basandosi sui loro stessi risultati.

3.1 Risultati per la previsione un passo in avanti

In questa sezione si valutano le capacità predittive dei modelli quando l'orizzonte di previsione è pari a 1. Si ricorda che per ciascun modello, nella prima iterazione, si considerano le 18 osservazioni iniziali per stimare il modello, utilizzato poi per prevedere la 19-esima osservazione. Successivamente la 19-esima osservazione entra nell'insieme di stima e si ripete la procedura fino alla previsione dell'ultimo valore disponibile (1000).

La **Figura 2** mostra il rango medio di ciascun modello sulle 90 serie temporali e la sua evoluzione con l'aumentare della dimensione del campione. L'asse delle ascisse rappresenta la dimensione dell'insieme di stima, sull'asse delle ordinate invece vi è il rango medio, ogni modello è rappresentato da una linea, di colore relativo al metodo

di appartenenza (apprendimento automatico o statistico). Le due linee più spesse sono una semplice media dei ranghi di tutti i modelli appartenenti al medesimo metodo. La linea nera verticale che cade nel punto 144, rappresenta la dimensione massima del campione nello studio di *Makridakis et al. 2018*. I risultati in figura evidenziano che quando la dimensione del campione è piccola, i metodi statistici ad eccezione del *naive stagionale*, presentano capacità predittive sistematicamente migliori rispetto ai metodi di apprendimento automatico. Tuttavia all'aumentare della dimensione dell'insieme di stima i metodi di apprendimento automatico li superano.

Figura 2: Curva di apprendimento per la previsione un passo in avanti utilizzando il rango medio di ogni metodo di previsione, lisciato con una media mobile di ordine 50, Cerqueira et al. 2019.

Nella **Figura 3** è presente la stessa analisi ma effettuata senza tener conto del modello *naive stagionale* poiché le sue scarse prestazioni hanno penalizzato i risultati

dei metodi statistici. Le conclusioni tratte rimangono invariate.

Figura 3: Curva di apprendimento per la previsione un passo in avanti utilizzando il rango medio di ciascun metodo di previsione, escluso il Naive stagionale, lisciato una media mobile di ordine 50, Cerqueira et al. 2019.

Infine la **Figura 4** porta l'analisi precedente in termini di *MASE*: notiamo che entrambi i metodi migliorano la loro capacità predittiva all'aumentare della dimensione del campione e confermano i risultati delle analisi effettuate sul metodo del rango.

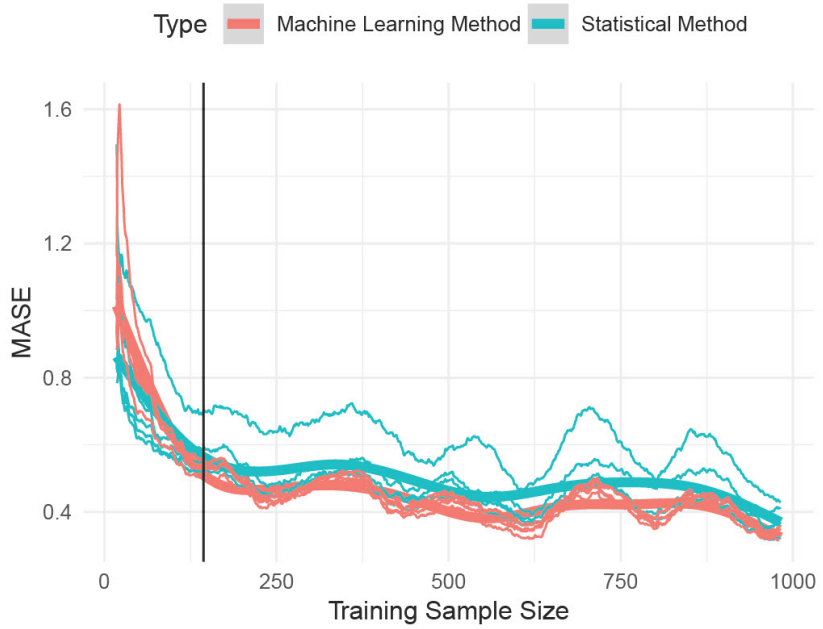

Figura 4: Curva di apprendimento per la previsione un passo in avanti utilizzando il MASE di ciascun metodo di previsione, lisciato con una media mobile di ordine 50, Cerqueira et al. 2019.

Nella **Tabella 1** riporto per ciascun modello la media della media mobile applicata sui ranghi.

RBT	SARIMA	ETS	RF	GP	TBATS	MARS	GLM	THETA	SNAIVE
5.11	5.26	5.39	5.39	5.39	5.45	5.47	5.54	5.73	6.28

Tabella 1: Media del rango medio di ciascun modello per la previsione un passo in avanti, Cerqueira et al. 2019.

Il modello che presenta il punteggio migliore è il *Rule-Based Tree* seguito dal *SARIMA*, si evince che non è sempre vero che un modello *machine learning* superi costantemente i modelli statisitci in termini di precisione delle previsioni e che invece può essere più performante (per esempio rispetto ai restanti 4 modelli di apprendimento automatico). Si nota anche che il modello *Random Forest* ha ottenuto una posizione peggiore rispetto al modello *Rule-Based Tree*, questo potrebbe dipendere sia dal

fatto che quest'ultimo imposta dei modelli di regressione sulle foglie e non produce come output una semplice media, sia perché le Foreste Casuali hanno la tendenza a fare *overfitting*: se il numero di alberi è elevato, diventando complesse e ridondanti, d'altra parte, se il numero di alberi è troppo basso, potrebbe non essere in grado di catturare tutti pattern presenti nella serie storica. Il modello con il punteggio peggiore come già annunciato è il *Naive stagionale* che non tiene conto di tendenze nella serie storica.

3.2 Risultati per la previsione più passi in avanti

L'approccio utilizzato è simile a quello per la previsione un passo in avanti, la differenza è che, invece di prevedere solo il valore successivo, vengono previste tutte le 18 osservazioni successive. Quindi al primo passo della procedura si stima il modello con le prime 18 osservazioni disponibili per prevedere le 18 successive (cioè dalle 19-esima alle 36-esima). Alla seconda iterazione l'insieme di addestramento è cresciuto aggiungendo la 19-esima osservazione, ora si ripete l'esercizio. La procedura di confronto tra i diversi modelli è illustrata nel paragrafo 2.3.

Le **Figure 5 e 6** mostrano la media (sulle 90 serie storiche) di ciascun modello man mano che il set di addestramento cresce secondo la procedura appena descritta. Le figure sono simili, ma i risultati del metodo *naive stagionale* sono stati esclusi dalla seconda, utilizzando la stessa logica di prima. Analizzando i risultati in base al tipo di modello, la principale conclusione in questo scenario è simile a quella ottenuta nell'impostazione un passo in avanti: in base alle linee smussate, i metodi statisti-

ci sono migliori di quelli di apprendimento automatico solo quando la dimensione del campione è piccola. In questo scenario, tuttavia, i due metodi sembrano pareggiarsi con l'aumentare della dimensione del campione di addestramento (a patto di ignorare il *naive stagionale*).

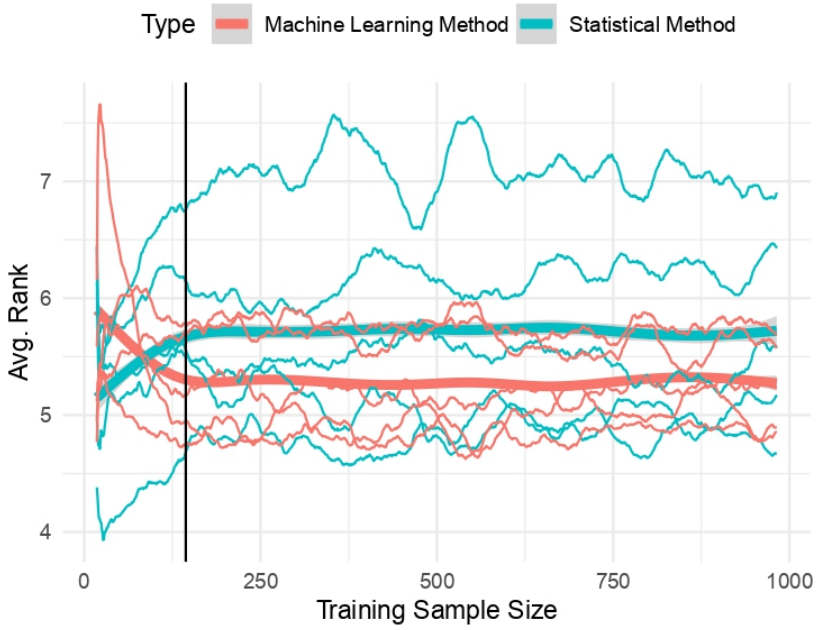

Figura 5: Curva di apprendimento per la previsione più passi in avanti utilizzando il rango medio lisciato di ciascun metodo di previsione per l'orizzonte previsivo pari a 18, Cerqueira et al. 2019.

Figura 6: Curva di apprendimento per la previsione più passi in avanti utilizzando il rango medio lisciato di ciascun metodo di previsione, escluso il *Naive stagionale*, per l'orizzonte previsivo pari a 18, Cerqueira et al. 2019.

Nella **Figura 7**, come potevamo aspettarci, notiamo che i punteggi *MASE* sono più alti rispetto alla previsione a un passo, questo è dovuto alla sottostante maggior incertezza nel fare previsioni con orizzonti di previsione più lunghi.

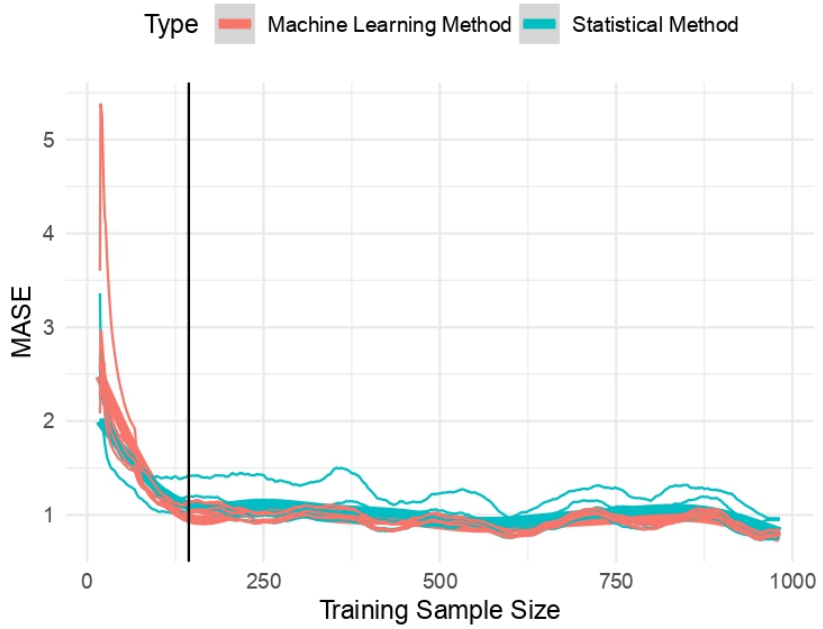

Figura 7: Curva di apprendimento per la previsione più passi in avanti utilizzando il MASE lisciato di ciascun metodo di previsione, per l'orizzonte previsivo pari a 18, Cerqueira et al. 2019.

La **Tabella 2** presenta i risultati per singolo modello in modo simile alla Tabella 1, ma per la previsione a più passi in avanti.

SARIMA	RBT	GLM	TBATS	GP	ETS	MARS	RF	THETA	SNAIVE
4.77	4.95	4.97	5.11	5.24	5.43	5.69	5.74	6.12	6.98

Tabella 2: Media del rango medio di ciascun modello per la previsione più passi in avanti, Cerqueira et al. 2019.

Il modello migliore per punteggio è il *SARIMA* che supera tutti i modelli di apprendimento automatico, in particolare per le prime 144 come si nota nella Figura 6. Rimangono nelle ultime due posizioni rispettivamente i modelli *THETA* e *Naive stagionale*: entrambi i modelli sono semplici, ad esempio il modello *THETA* assume che la serie storica abbia una tendenza costante nel tempo. Queste semplificazioni implicano una complessità

computazionale ridotta a fronte di una limitata capacità previsiva.

3.3 Complessità computazionale

La complessità computazione (CC) di un modello m, in questo caso, rappresenta una misura del tempo impiegato dal modello per completare la procedura prequenziale normalizzata col tempo impiegato dal modello *naive stagionale*. Viene definita come:

$$CC = \frac{Tempo\ impiegato\ dal\ modello\ m}{Tempo\ impiegato\ da\ snaive} \tag{3.1}$$

I risultati sono mostrati nella **Figura 8** come grafico a barre scalato con una trasformazione logaritmica, ogni colonna riporta i valori non scalati, da cui si evince che il modello computazionalmente più complesso è l'*ARIMA stagionale* mentre il migliore (dopo il *naive stagionale*) è il modello *THETA*. Notiamo un grande salto tra i valori riguardanti i primi due modelli e i restanti, questo è dovuto al fatto che il secondo gruppo necessita di operazioni di ottimizzazione dei parametri.

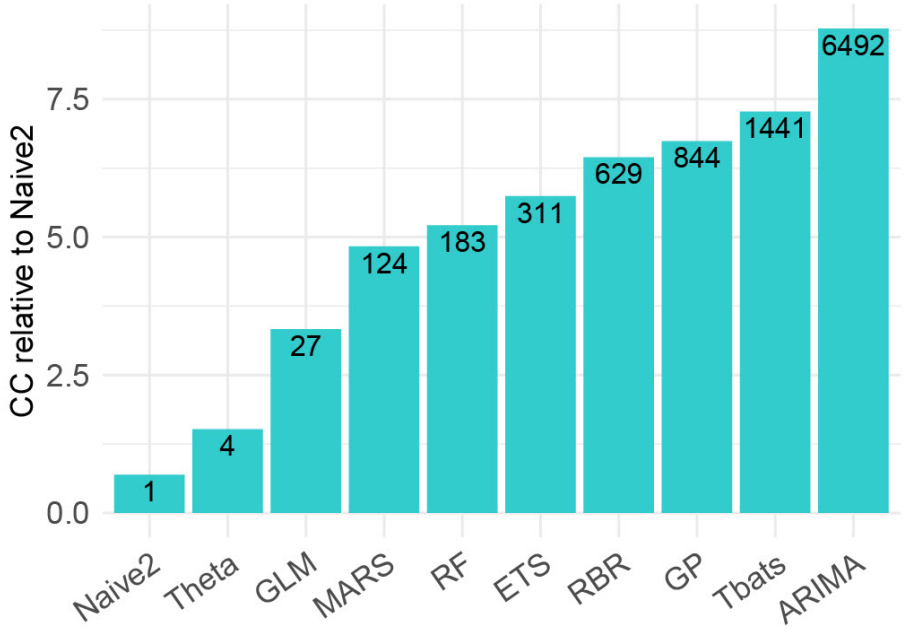

Figura 8: Complessità computazionale di ciascun metodo rispetto al modello di riferimento Naive2 (in scala logaritmica), Cerqueira et al. 2019.

Capitolo 4

Replicazione sul secondo insieme di serie storiche

Questo capitolo tratta i risultati ottenuti dall'applicazione dell'esperimento di *Cerqueira et al. (2019)* su dati differenti da quelli considerati dagli autori, per la descrizione si rimanda al paragrafo 2.1. Le serie storiche sono tagliate a 500 osservazioni poiché dai risultati del *capitolo 3* sembra sufficiente svolgere l'analisi su un numero inferiore di osservazioni.

4.1 Risultati per la previsione un passo in avanti

Di seguito si riportano i risultati dell'esperimento quando si considera l'orizzonte previsivo pari a 1. La procedura è identica a quella della sezione 3.1.

La **Figura 9** porta alle medesime conclusioni per cui all'aumentare della dimensione del campione, i metodi di apprendimento automatico superano quelli statistici. In queste circostanze addirittura bastano poco meno di 100 osservazioni per notare la differenza tra i due metodi ed è particolarmente marcata quando il modello viene addestrato sull'insieme completo.

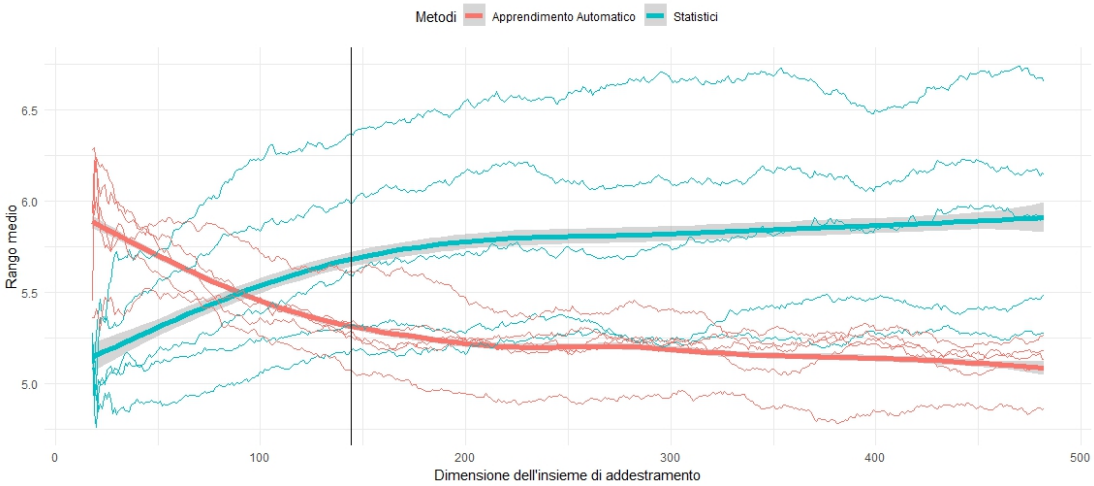

Figura 9: Curva di apprendimento per la previsione un passo in avanti utilizzando il rango medio di ogni metodo di previsione, lisciato con una media mobile di ordine 50.

Anche in questo caso il modello *naive stagionale* ha delle performance costantemente peggiori di tutti gli altri perciò viene rimosso. I risultati nella **Figura 10** non portano a conclusioni differenti dalle attuali.

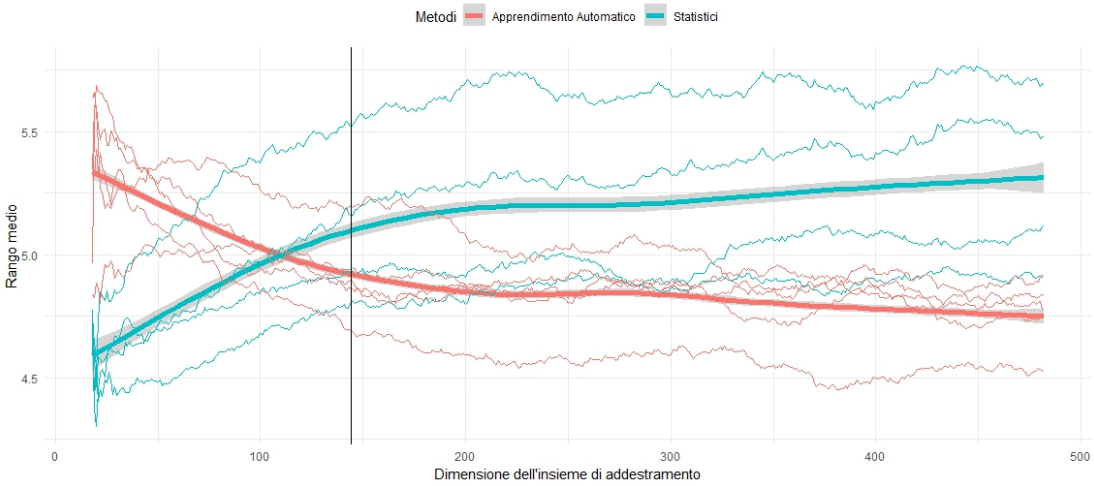

Figura 10: Curva di apprendimento per la previsione un passo in avanti utilizzando il rango medio di ciascun metodo di previsione, escluso il Naive stagionale, lisciato una media mobile di ordine 50.

La **Figura 11** che mostra i valori lisciati del *MASE*

all'aumentare della dimensione del campione conferma che i metodi statistici hanno performance previsive peggiori di quelle dei metodi di apprendimento automatico. In particolare dopo le 150 osservazioni circa, i metodi statistici sono costantemente peggiori rispetto agli altri (a parte la metodologia *Random Forest*).

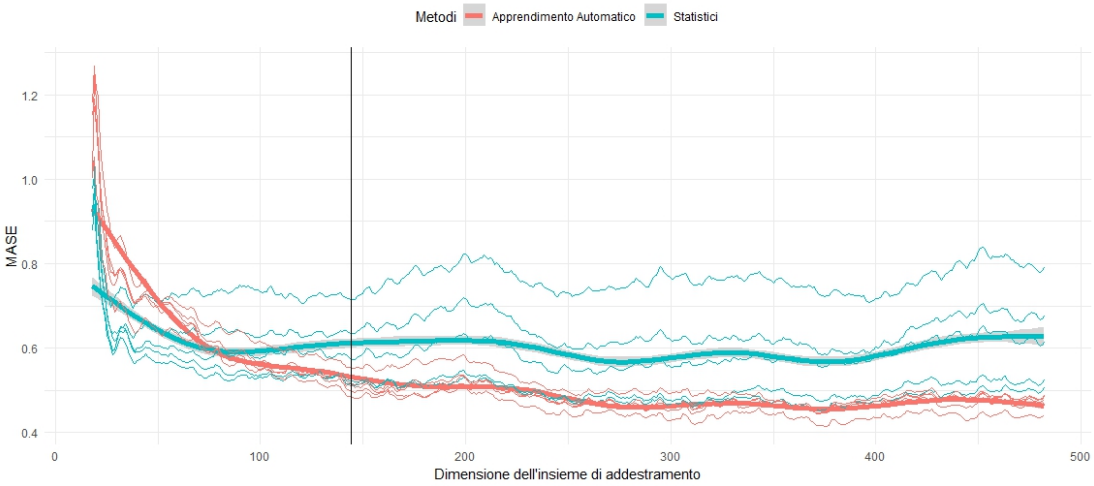

Figura 11: Curva di apprendimento per la previsione un passo in avanti utilizzando il MASE di ciascun metodo di previsione, lisciato con una media mobile di ordine 50.

Infine la **Tabella 3** riporta il valore medio per ciascun modello della media mobile applicata ai ranghi.

RBT	SARIMA	GP	TBATS	GLM	MARS	RF	ETS	THETA	SNAIVE
5.05	5.7	5.25	5.32	5.32	5.34	5.46	5.66	6.01	6.43

Tabella 3: Media del rango medio di ciascun modello per la previsione un passo in avanti.

I valori sono simili a quelli della Tabella 1 però vi è un forte peggioramento del modello *ETS*, dovuto al fatto che il modello guadagna in termini di previsione, all'aumentare dell'insieme di stima, di più rispetto agli altri metodi e considerando solo serie storiche di lunghezza 500 viene penalizzato. Anche in questa situazione i modelli

THETA e *Naive stagionale* hanno prestazioni previsive peggiori rispetto a tutti gli altri metodi.

4.2 Risultati per la previsione più passi in avanti

Come per il paragrafo precedente, si fa riferimento al paragrafo 3.2 circa i dettagli sulla procedura.

Nelle **Figure 12 e 13** mostro la media di ciascun modello quando l'insieme di addestramento cresce, nel secondo caso non viene considerato il modello *naive stagionale*. In entrambe le figure, le due linee, quasi dall'inizio evidenziano come i modelli di apprendimento automatico riescano a modellare meglio le serie storiche per prevedere i valori futuri a più passi in avanti. La differenza tra i due metodi è crescente in tutta la sua evoluzione, non appare alcuna tendenza ad annullarsi come nelle figure 5 e 6.

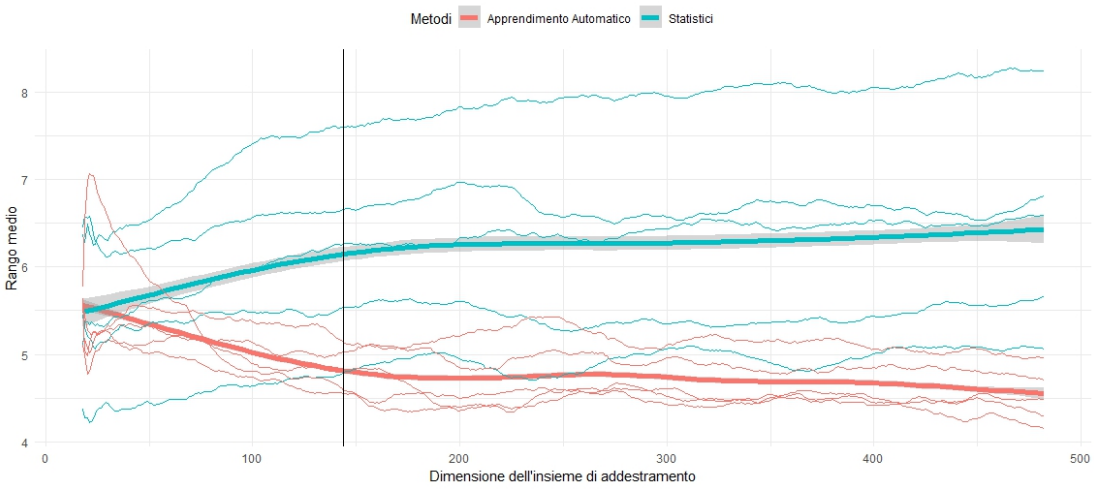

Figura 12: Curva di apprendimento per la previsione più passi in avanti utilizzando il rango medio lisciato di ciascun metodo di previsione per l'orizzonte previsivo pari a 18.

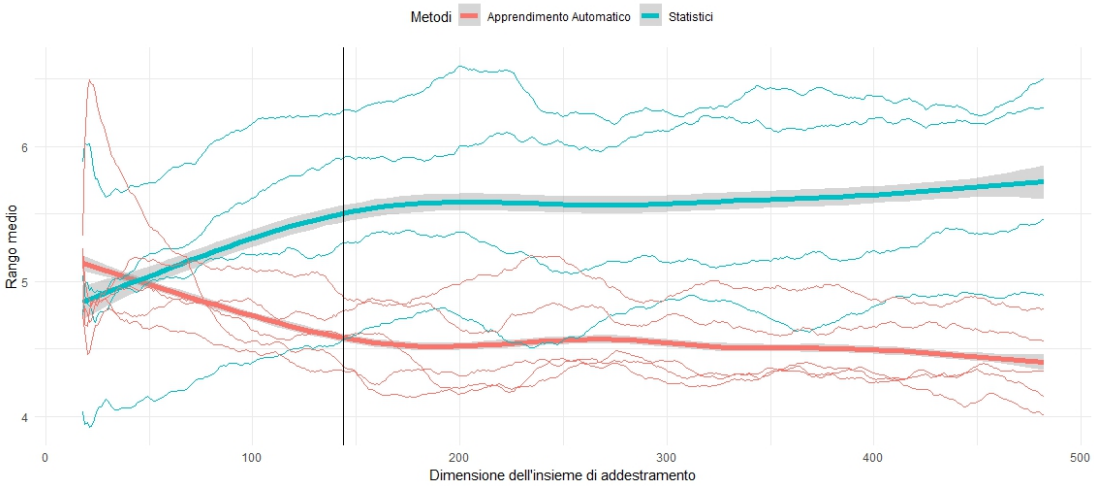

Figura 13: Curva di apprendimento per la previsione più passi in avanti utilizzando il rango medio lisciato di ciascun metodo di previsione, escluso il *Naive stagionale*, per l'orizzonte previsivo pari a 18.

La **Figura 14** riporta i valori lisciati del *MASE* per ciascun modello. I modelli statistici sembrano peggiorare le loro capacità predittive all'aumentare della dimensione del campione.

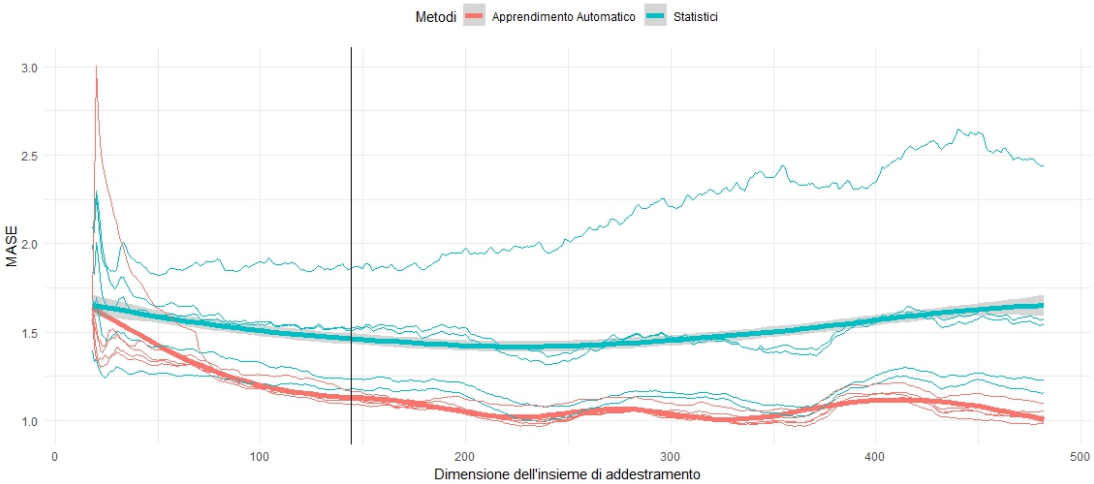

Figura 14: Curva di apprendimento per la previsione più passi in avanti utilizzando il rango medio lisciato di ciascun metodo di previsione, escluso il *Naive stagionale*, per l'orizzonte previsivo pari a 18.

Infine si riportano i valori medi per ogni modello del-

la media mobile applicata ai ranghi nella **Tabella 4**. Si nota lo "scambio di posizione" del modello *SARIMA* a favore dei *Processi Gaussiani (GP)* che occupano la posizione migliore.

GP	RBT	GLM	SARIMA	RF	MARS	TBATS	ETS	THETA	SNAIVE
4.59	4.66	4.71	4.85	4.95	5.21	5.44	6.25	6.62	7.72

Tabella 4: Media del rango medio di ciascun modello per la previsione più passi in avanti.

4.3 Complessità computazionale

La complessità computazionale dei modelli, definita nel paragrafo 3.3, è illustrata nella **Figura 15**. In questa caso, i modelli statistici (senza considerare il *TBATS* che è il peggiore) sono computazionalmente meno complessi rispetto a tutti i modelli di apprendimento automatico (tranne per i *GLM* che supera il modello *SARIMA*).

Il modello *ARIMA Stagionale* guadagna posizioni paragonato agli altri modelli, rispetto alla figura 8 dalla replicazione dell'esperimento. Quindi il tempo che impiega *auto.arima()* per le stime dei parametri è buono per le prime 500 osservazioni, peggiora drasticamente per le successive. La considerazione effettuata assume che le serie storiche dei due esperimenti abbiano strutture sottostanti simili, altrimenti uno dei due è stato penalizzato dalla complessità strutturale delle serie storiche.

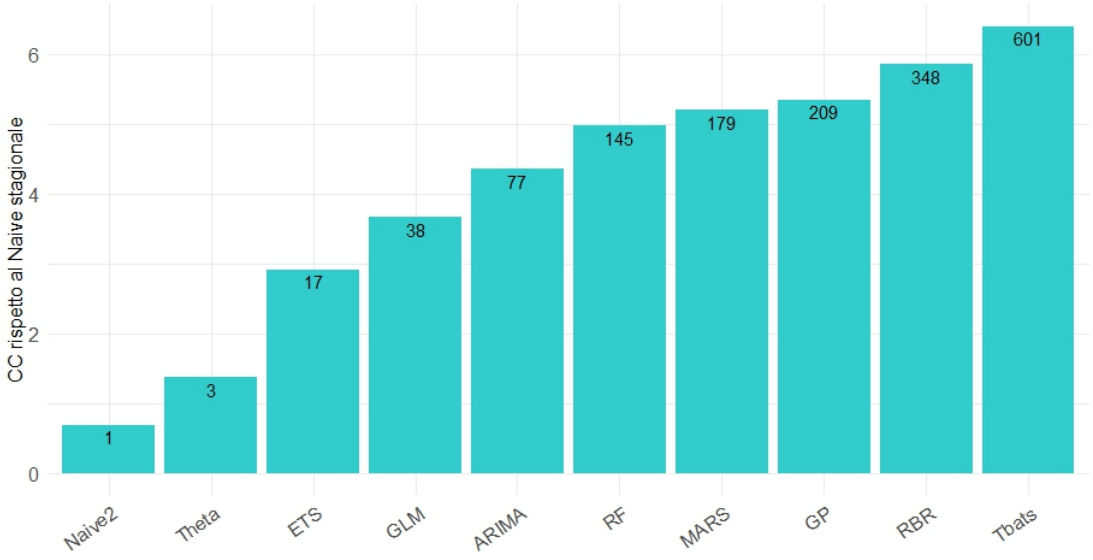

Figura 15: Complessità computazionale di ciascun metodo rispetto al modello di riferimento Naive2 (in scala logaritmica).

Conclusioni

L'analisi svolta sulle 90 (e poi sulle 80) serie storiche prese in considerazione è conforme con le aspettative iniziali tuttavia ci sono delle considerazioni da tenere presente.

Una di queste è che i metodi di apprendimento automatico nelle previsioni a più passi in avanti non hanno un guadagno in termini di prestazioni predittive all'aumentare delnumero di osservazioni, questo potrebbe dipendere dalla necessità di allungare le serie storiche oltre le 1000 osservazioni per permettere una stima più accurata. È difficile valutare a priori la dimensione ottimale, ma insieme ad un controllo più stringente sulla struttura dei dati, può essere spunto per analisi future.

Inoltre, si fa presente che i modelli di apprendimento automatico non sono in grado di generalizzare da insiemi di piccole dimensioni, ed è un limite rispetto ai modelli statistici.

Un altro aspetto riguarda la relazione tra il costo computazionale e la scelta del modello. In generale l'attenzione non può essere volta solo alla capacità previsiva, infatti a fronte di grandi moli di dati potrei scegliere un compromesso con il costo, sia in termini di tempo che di risorse. In base all'obbiettivo e alle risorse, potrei preferire un modello interpretabile e che mi permetta di generalizzare i risultati rispetto ad un modello *black-box* (o di intricata interpretabilità) che presenta miglior precisione nelle previsioni ma un costo computazionale ele-

vato e per i quali non è immediatamente evidente quali caratteristiche siano più importanti per le previsioni o come hanno combinato le informazioni per ottenere un risultato specifico.

Infine, osserviamo che, anche con una grande quantità di dati, non è ovvio che un metodo di apprendimento automatico superi un metodo statistico. Questo ragionamento è in accordo con il teorema *No Free Lunch* (Wolpert 1996), che afferma che nessun algoritmo di apprendimento è il più appropriato in tutti gli scenari.

Appendice A

Comandi R per analisi

A.1 Creazione del dataset

Di seguito le operazioni per l'aggregazione, la pulizia e qualche controllo sulle serie storiche utilizzate per l'analisi.

```
rm(list=ls())
library(tsdl)
library(forecast)
library(fpp2)
library(ggplot2)
library(datasets)
library(astsa)

#----------
#serie storiche in TSA
files1 <- list.files(path = "ts_TSA",
                     pattern = "*.dat",
                     full.names = T)

data1 <- lapply(files1, read.table)

#considero le serie storiche di lunghezza almeno 500
lung <- c()
for(i in 1:length(data1)){
  if(nrow(data1[[i]]) >= 500) lung = c(lung, i)
}

data <- list()

for(i in lung){
  elem = subset(data1[[i]])[2:501,]
  data <- c(data, list(elem))
}
#9 SS

#----------
#serie storiche in forecast

data(package='forecast')$results
#datasets in forecast: gas,gold,taylor,wineind,woolyrnq
data2 <- list(gas,gold,taylor,wineind,woolyrnq)

lung <- c()
for(i in 1:length(data2)){
  if(length(data2[[i]]) >= 500) lung = c(lung, i)
}
data2[[2]][which(is.na(data2[[2]]))] <- mean(data2[[2]][1:500], na.rm = T)

data <- c(data, list(data2[[2]][1:500]), list(data2[[3]][1:500]))
#2 SS (11 in totale)

#----------
#serie storiche in fpp2

data(package = "fpp2")$results[,3]
data3 <- c(list(a10), list(arrivals), list(ausair), list(ausbeer), list(auscafe), list(austa),
           list(austourists), list(calls), list(debitcards), list(departures), list(elecdaily),
           list(elecdemand), list(elecequip), list(elecsales), list(euretail), list(gasoline),
           list(goog), list(goog200), list(guinearice), list(h02), list(hyndsight), list(prison),
           list(insurance), list(livestock), list(marathon), list(maxtemp), list(melsyd), list(mens400),
           list(oil), list(prisonLF), list(qauselec), list(qcement), list(qgas), list(sunspotarea),
           list(uschange), list(usmelec), list(visnights), list(wmurders))
```

```
lung <- c()
for(i in 1:length(data3)){
  if(length(data3[[i]]) >= 500) lung = c(lung, i)
}
data3[[27]][which(is.na(data3[[27]]))] <- mean(data3[[27]][1:500], na.rm = T)

for(i in lung){
  data <- c(data, list(data3[[i]][1:500]))
}
##11 SS (22 in totale)

#----------
#ss in ggplot2
nrow(economics)
ncol(economics)

for(i in 2:6){
  data <- c(data, list(economics[1:500,i]))
}
#5 SS (27 in totale)

#----------
#ss in datasets
ts1 <- datasets::EuStockMarkets[1:500]
data <- c(data, list(ts1))
#1 SS (28 in totale)

#----------
#ts in astsa

library(astsa)

series_names <- ls("package:astsa")
series_lengths <- numeric(length(series_names))

for (i in seq_along(series_names)) {
  series_name <- series_names[i]
  series_data <- get(series_name)
  if (is.ts(series_data)) {
    series_lengths[i] <- length(series_data)
  }
}
series_names <- series_names[which(series_lengths>=500)]

data4 <- c(list(arf[1:500]), list(BCJ[1:500]), list(bnrf1ebv[1:500]), list(bnrf1hvs[1:500]),
           list(cardox[1:500]), list(cmort[1:500]), list(econ5[1:500]), list(ENSO[1:500]),
           list(EQ5[1:500]), list(EXP6[1:500]), list(fmri1[1:500]),
           list(lap[1:500]), list(nyse[1:500]), list(part[1:500]),
           list(so2[1:500]), list(sp500.gr[1:500]), list(speech[1:500]), list(star[1:500]),
           list(tempr[1:500]), list(UnempRate[1:500]), list(varve[1:500]))

data <- c(data, data4)
#21 SS (49 in totale)

#----------

tsdl_list <- tsdl[sapply(tsdl, length) > 500 & sapply(tsdl, length) < 1000]
tsdl_list <-
  tsdl_list[!sapply(tsdl_list,
                    function(x) {
                      any(is.na(x))
                    })]

tsdl_list[] <- lapply(tsdl_list, head, 500)

is_univar <- sapply(tsdl_list, class) == "ts"

tsdl_list <- tsdl_list[is_univar]
tsdl_list <- tsdl_list[1:30]

data <- c(data, tsdl_list)
#84 SS (80 in totale)

#----------

rm(data1, data2, data3, data4, tsdl_list, elem, files1, i, lung, ts1, is_univar,
   series_lengths, series_name, series_names)

#dati mancanti?
for(i in 1:length(data)){
  if(sum(is.na(data[[i]])) != 0) print(paste('there-is-NA-in-ts-number', i))
}

for(i in 1:length(data)){
  data[[i]] = as.matrix(as.numeric(as.matrix(data[[i]])))
}
#----------

ts_list <- data
rm(data)
#

tdiff <-
  sapply(ts_list,
         function(x) {
           xt <-
             tryCatch(
```

```
                as.POSIXct(head(rownames(as.data.frame(x)))),
                error = function(e)
                  NA
              )

            difftime(xt[2],xt[1], units = "hours")
          })

#lapply(ts_list[is.na(tdiff)], head, 3)
tdiff[is.na(tdiff)] <- 1

FRQs <- tdiff
FRQs[tdiff == 1] <- 24
FRQs[tdiff == .5] <- 48
FRQs[tdiff == 24] <- 365

min_len <- 500

ts_list <-
  lapply(1:length(ts_list),
         function(i) {
           x <- ts_list[[i]]
           x <- ts(as.vector(x), frequency = FRQs[i])
           x <- head(x, min_len)
           x
         })

#Controllo se ci sono duplicati
list_identical <- c()
nl <- 1:length(ts_list)

for(i in nl){
  for(j in nl[i])
    if(identical(ts_list[[i]], ts_list[[j]])) list_identical = rbind(list_identical, c(i,j))
}
sum(apply(list_identical, 1, diff))

#sistemo il formato
ciclconv <- 1:length(ts_list)
for(i in ciclconv)
  ts_list[[i]] <- ts(matrix(ts_list[[i]], ncol = 1))
#
save(ts_list, file = "ts_data.rdata")
```

A.2 Funzioni per la stima dei modelli

A.2.1 Modelli statistici

La funzione stat_prediction prende come argomenti i dati di addestramento (train), l'orizzonte di previsione (h) e il tipo di modello (modeltype). In base al modello applica le relative funzioni di stima per restituire i valori previsti e il tempo di esecuzione.

```
stat_prediction <-
  function(train,h,modeltype) {
    t0 <- Sys.time()

    modelf <-
      switch(modeltype,
             "arima" = arima_model,
             "naive" = naive_model,
             "ets" = ets_model,
             "tbats" = tbats_model,
             "theta" = theta_model,
             naive_model)

    yh <- modelf(train, h)

    t1 <- Sys.time() - t0

    list(y_hat=yh,time=t1)
  }

#ARIMA Stagionale
arima_model <-
  function(train,h) {
    require(forecast)

    model <- tryCatch(
      forecast::auto.arima(y = train),
      error = function(e)
        naive(train,h=h)
```

```
    )

    yh <- forecast::forecast(model, h=h)

    as.vector(yh$mean)
  }

#Exponential Smoothing State Space
ets_model <-
  function(train,h) {
    require(forecast)

    model <- tryCatch(
      forecast::ets(y = train),
      error = function(e)
        naive(train,h=h)
    )

    yh <- forecast::forecast(model, h=h)

    as.vector(yh$mean)
  }

#Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend, and Seasonal components
tbats_model <-
  function(train,h) {
    require(forecast)

    model <- tryCatch(
      forecast::tbats(y = train),
      error = function(e)
        naive(train,h=h)
    )

    yh <- forecast::forecast(model, h=h)

    as.vector(yh$mean)
  }

#NAIVE Stagionale
naive_model <-
  function(train,h) {
    require(forecast)

    yh <- snaive(train,h=h)

    as.vector(yh$mean)
  }

#THETA
theta_model <-
  function(train,h) {
    require(forecast)

    yh <- tryCatch(thetaf(y = train,h=h),
                   error = function(e) {
                     naive(train,h=h)
                   })

    as.vector(yh$mean)
  }}
```

A.2.2 Modelli di apprendimento automatico

Il codice rappresenta l'analogo del precedente però per i metodi di apprendimento automatico. La funzione train_prediction prende come argomenti la formula che specifica la relazione tra le variabili (form), i dati di addestramento (train), l'orizzonte previsivo (h) e il tipo di modello (modeltype). All'interno della funzione, nell'oggetto "complete_par_list, si specificano alcuni parametri necessari per ciascun modello, che saranno andati a testare in fase di stima. Anche in questa situazione si tiene traccia del tempo di esecuzione per ciascun metodo che verrà resti-

tuito al termine assieme al vettore y_hat delle previsioni per ogni valore dell'orizzonte previsivo.

```
train_prediction <-
  function(form,train,h,modeltype) {

    #parametri richiesti dai modelli
    complete_par_list <-
      list(rf= list(num.trees = c(50,100,250,500)),
           gprocess = list(kernel = c("rbfdot","vanilladot", "polydot","laplacedot"),
                           tolerance=c(0.01,0.001)),
           rbr = list(committees = c(1, 5, 10, 25, 50)),
           lasso = list(alpha = c(0,.25,.5,.75,1)),
           mars = list(pmethod = c("forward","backward"),
                       degree = c(1,2,3),
                       nk =c(2,5,7,15))
      )

    t0 <- Sys.time()

    m <- tryCatch(train_model(form,train,modeltype, complete_par_list[[modeltype]]),
                  error = function(e) {
                    modeltype <<- "rf"
                    train_model(form,train,modeltype, complete_par_list[[modeltype]])
                  })


    test <- train[nrow(train),]
    test[,2:ncol(test)] <- unlist(test[,1:(ncol(test)-1)])
    test[,"target"] <- -1

    yh <- numeric(h)
    for (i in 1:h) {
      yh[i] <- predict_model(m,test,modeltype,form)

      if (is.na(yh[i])) {
        yh[i] <- mean(unlist(test[,-1]))
      }

      test[,3:ncol(test)] <- test[,2:(ncol(test)-1)]
      test$Tm1 <- yh[i]
    }

    names(yh) <- paste0("h_",1:h)

    t1 <- Sys.time() - t0

    list(y_hat=yh,time=t1)
  }

train_model <-
  function(form,train,modeltype, par_list) {

    model <-
      switch(modeltype,
             "rf" = lrandomforest(form, train,par_list),
             "gprocess" = gprocess(form, train,par_list),
             "rbr" = rbr(form, train,par_list),
             "lasso" = lasso(form, train,par_list),
             "mars" = mars(form, train,par_list),
             lrandomforest(form, train,par_list))
  }

predict_model <-
  function(model,newdata,modeltype,form) {
    newX <- model.matrix(form, newdata)
    yh <-
      switch(modeltype,
             "rf" = predict(model, newdata)$predictions,
             "gprocess" = predict(model,newdata)[,1],
             "mars" = predict(model,newdata)[,1],
             "rbr" = predict(model,newX),
             "lasso" = unname(predict(model,newX)[,1]),
             predict(model, newdata)$predictions)

    yh
  }

#Random Forest
lrandomforest <-
  function(form,train, par_list) {
    require(ranger)
    mtry_ <- floor(ncol(train) / 3)

    in_tr_ts <- holdout_(train, .9)
    in_tr <- in_tr_ts$train
    in_ts <- in_tr_ts$test
    in_y <- get_y(in_ts, form)

    exp_pl <- expand.grid(par_list)

    seq. <- 1:nrow(exp_pl)

    in_results <- numeric(nrow(exp_pl))
    for (i in seq.) {
      in_m <- ranger(form,in_tr,
```

```
                        num.trees = exp_pl[i,"num.trees"],
                        mtry = mtry_)

      in_yh <- predict_model(in_m, in_ts, "rf", target ~.)
      in_results[i] <- mean(smape_cal(in_y, in_yh))
    }

    best_var <- which.min(in_results)
    if (length(best_var) < 1) best_var <- 4

    ranger(form,train,
           num.trees = exp_pl[best_var,"num.trees"],
           mtry = mtry_)
  }

Processi Gaussiani
gprocess <-
  function(form,train, par_list) {
    require(kernlab)

    in_tr_ts <- holdout_(train, .9)
    in_tr <- in_tr_ts$train
    in_ts <- in_tr_ts$test
    in_y <- get_y(in_ts, form)

    exp_pl <- expand.grid(par_list)

    seq. <- 1:nrow(exp_pl)

    in_results <- numeric(nrow(exp_pl))
    for (i in seq.) {
      in_m <- gausspr(form,in_tr,
                      type = "regression",
                      kernel=as.character(exp_pl[i,"kernel"]),
                      tol=exp_pl[i,"tolerance"])

      in_yh <- predict_model(in_m, in_ts, "gprocess", target ~.)
      in_results[i] <- mean(smape_cal(in_y, in_yh))
    }

    best_var <- which.min(in_results)
    if (length(best_var) < 1) best_var <- 4

    gausspr(form,train,
            type = "regression",
            kernel=as.character(exp_pl[best_var,"kernel"]),
            tol=exp_pl[best_var,"tolerance"])
  }

#Rule-Based Tree
rbr <-
  function(form,train, par_list) {
    require(Cubist)

    in_tr_ts <- holdout_(train, .9)
    in_tr <- in_tr_ts$train
    in_ts <- in_tr_ts$test
    in_y <- get_y(in_ts, form)

    exp_pl <- expand.grid(par_list)

    seq. <- 1:nrow(exp_pl)

    in_results <- numeric(nrow(exp_pl))
    for (i in seq.) {
      in_X <- stats::model.matrix(form, in_tr)
      in_Y <- get_y(in_tr, form)

      in_m <- cubist(in_X, in_Y,
                     committees = exp_pl[i,"committees"])

      in_yh <- predict_model(in_m, in_ts, "rbr", target ~.)
      in_results[i] <- mean(smape_cal(in_y, in_yh))
    }

    best_var <- which.min(in_results)
    if (length(best_var) < 1) best_var <- 4

    X <- stats::model.matrix(form, train)
    Y <- get_y(train, form)

    cubist(X, Y,
           committees = exp_pl[best_var,"committees"])
  }

#LASSO/RIDGE/ELASTIC NET
lasso <-
  function(form,train, par_list) {
    require(glmnet)

    in_tr_ts <- holdout_(train, .9)
    in_tr <- in_tr_ts$train
    in_ts <- in_tr_ts$test
    in_y <- get_y(in_ts, form)

    exp_pl <- expand.grid(par_list)

    seq. <- 1:nrow(exp_pl)
```

```
    in_results <- numeric(nrow(exp_pl))
    for (i in seq.) {
      in_X <- stats::model.matrix(form, in_tr)
      in_Y <- get_y(in_tr, form)

      m.all <- glmnet(in_X, in_Y,
                      alpha = exp_pl[i,"alpha"],
                      family = "gaussian")

      in_m <- glmnet(in_X,in_Y,
        alpha = exp_pl[i,"alpha"],
        lambda = min(m.all$lambda),
        family = "gaussian")

      in_yh <- predict_model(in_m, in_ts, "lasso", target ~.)
      in_results[i] <- mean(smape_cal(in_y, in_yh))
    }

    best_var <- which.min(in_results)
    if (length(best_var) < 1) best_var <- 4

    X <- stats::model.matrix(form, train)
    Y <- get_y(train, form)

    m.all <- glmnet(X, Y,
                    alpha = exp_pl[best_var,"alpha"],
                    family = "gaussian")

    glmnet(
      X,
      Y,
      alpha = exp_pl[best_var, "alpha"],
      lambda = min(m.all$lambda),
      family = "gaussian"
    )
  }

#Multivariate Adaptive Regression Splines
mars <-
  function(form,train, par_list) {
    require(earth)

    in_tr_ts <- holdout_(train, .9)
    in_tr <- in_tr_ts$train
    in_ts <- in_tr_ts$test
    in_y <- get_y(in_ts, form)

    exp_pl <- expand.grid(par_list)

    seq. <- 1:nrow(exp_pl)

    in_results <- numeric(nrow(exp_pl))
    for (i in seq.) {
      in_m <- earth(form,in_tr,
                    pmethod = as.character(exp_pl[i,"pmethod"]),
                    degree = exp_pl[i,"degree"],
                    nk = exp_pl[i,"nk"])

      in_yh <- predict_model(in_m, in_ts, "mars", target ~.)
      in_results[i] <- mean(smape_cal(in_y, in_yh))
    }

    best_var <- which.min(in_results)
    if (length(best_var) < 1) best_var <- 4

    earth(form,train,
          pmethod = as.character(exp_pl[best_var,"pmethod"]),
          degree = exp_pl[best_var,"degree"],
          nk = exp_pl[best_var,"nk"])
  }
```

A.2.3 Elaborazione preliminare

Di seguito presento due funzioni, utili per la pre-elaborazione dei dati, centrali per l'analisi. La funzione preprocessing_timeseries prende come argomenti i dati di addestramento (tr), un insieme di dati di test (tst), la frequenza della serie storica (frq) e un parametro (k). La funzione trasforma i dati nel formato di serie storiche, applica la trasformazione di Box-Cox, verifica la presenza di stagionalità (test di Cox-Stuart) e se presente destagionalizza

la serie. Invece la funzione apply_forecasting prende i dati generati dalla funzione precedente e definisce la formula per un modello di regressione e per ogni modello, richiama la funzione train_prediction che restituisce le previsioni. Qualora fossero state applicate operazioni di destagionalizzazione, si applica la funzione inversa.

```
  apply_forecasting <-
function(train_adj,
         train_adj_xi,
         train,
         train_xi,
         embedded_tr,
         seasonal_factor,
         lambda,
         h) {

  form <- target ~.

  modeltypes <-
    c("rf",
      "gprocess",
      "rbr",
      "mars",
      "lasso")

  yh_ml <-
    lapply(modeltypes,
           function(nm) {
             cat(nm,"\n")
             train_prediction(
               form = form,
               train = embedded_tr,
               h = h,
               modeltype = nm)
           })

  names(yh_ml) <- modeltypes

  statmtype1 = c("tbats","theta","naive")
  yh_st1 <-
    lapply(statmtype1,
           function(nm) {
             cat(nm,"\n")
             stat_prediction(train_adj, h, nm)
           })
  names(yh_st1) <- statmtype1

  statmtype2 <- c("arima","ets")
  yh_st2 <-
    lapply(statmtype2,
           function(nm) {
             cat(nm,"\n")
             stat_prediction(train, h, nm)
           })
  names(yh_st2) <- statmtype2

  if (!is.null(train_adj_xi)) {
    yh_ml <-
      lapply(yh_ml,
             function(x) {
               x$y_hat <- diffinv(x$y_hat,xi = train_adj_xi)[-1]
               x
             })

    yh_st1 <-
      lapply(yh_st1,
             function(x) {
               x$y_hat <- diffinv(x$y_hat,xi = train_adj_xi)[-1]
               x
             })
  }

  if (!is.null(seasonal_factor)) {
    yh_ml <-
      lapply(yh_ml,
             function(x) {
               x$y_hat <- x$y_hat * seasonal_factor
               x
             })

    yh_st1 <-
      lapply(yh_st1,
             function(x) {
               x$y_hat <- x$y_hat * seasonal_factor
               x
             })
  }

  if (!is.null(train_xi)) {
    yh_st2 <-
      lapply(yh_st2,
```

```
                function(x) {
                  x$y_hat <- diffinv(x$y_hat,xi = train_xi)[-1]
                  x
                })
    }

    Res <- c(yh_st2, yh_st1, yh_ml)

    Res <-
      lapply(Res,
             function(x) {
               x$y_hat <- InvBoxCox(x$y_hat, lambda)
               x
             })

    Y_hat <- lapply(Res, function(x) x$y_hat)
    Time <- lapply(Res, function(x) x$time)
    Time <- sapply(Time, calc_time)

    list(Y_hat=Y_hat, Time=Time)
  }


preprocessing_timeseries <-
  function(tr, tst, frq, k) {
    l <- length(tst)

    tr <- ts(tr, frequency = frq)
    tst <- ts(tst, frequency = frq)

    lambda <- BoxCox.lambda(tr)
    tr <- BoxCox(tr, lambda)
    tst <- BoxCox(tst, lambda)

    seas_test <- tryCatch(nsdiffs(tr), error = function(e) 0)
    if (seas_test > 0) {
      decomp <-
        tryCatch(decompose(tr, "multiplicative"),
                 error=function(e) {
                   NULL
                 })

      if (!is.null(decomp)) {
        tr_adj = tr / decomp$seasonal

        seasonal_factor <- rep(tail(decomp$seasonal, frq), trunc(1 + 1/frq))[1:1]

        tr_adj <- ts(tr_adj, frequency = frq)
      } else {
        tr_adj <- tr
        seasonal_factor <- NULL
      }
    } else {
      tr_adj <- tr
      seasonal_factor <- NULL
    }

    tr_adj[is.infinite(tr_adj)] <- NaN
    tr_adj[is.na(tr_adj)] <- mean(tr_adj, na.rm=TRUE)

    tr[is.infinite(tr)] <- NaN
    tr[is.na(tr)] <- mean(tr, na.rm=TRUE)

    coxstuart <- cox.stuart.test(tr_adj)$p.value
    if (coxstuart < .05) {
      tr_adj_xi <- tr_adj[length(tr_adj)]
      tr_adj <- diff(tr_adj, 1)
    } else {
      tr_adj_xi <- NULL
    }

    coxstuart2 <- cox.stuart.test(tr)$p.value
    if (coxstuart2 < .05) {
      tr_xi <- tr[length(tr)]
      tr <- diff(tr, 1)
    } else {
      tr_xi <- NULL
    }

    trk <- embed_timeseries_t(as.numeric(tr_adj), k)

    list(
      embedded_tr = trk,
      train_adj = tr_adj,
      train = tr,
      train_xi = tr_xi,
      train_adj_xi = tr_adj_xi,
      test = tst,
      seasonal_factor = seasonal_factor,
      lambda = lambda
    )
  }

embed_timeseries_t <-
  function(x, k) {
    tryCatch(embed_timeseries(x,k),
             error = function(e) {
               embed_timeseries(x,k/2)
             })
```

```
}
```

A.3 Previsioni

In questa sezione riporto il corpo centrale del codice: per ogni serie storica presente in "ts_list" parte dalle prime 18 osservazioni ed effettua la stima del modello e la previsione per la 19esima osservazione e per le osservazioni da 19 a 36, poi aggiorna l'insieme di stima e ripete fino all'ultimo valore possibile. Una volta terminato il procedimento per la prima serie storica, passa alla seconda e così via fino all'ultima. Salva i risultati nel file "EXPRES" che è una lista di lunghezza pari al numero delle serie storiche. Ogni elemento della lista è a sua volta una lista di lunghezza 500 che possiede per ogni suo elemento le previsioni per ogni modello e il tempo impiegato per la stima.

```
#DEFINISCO LA FUNZIONE PER L'ELABORAZIONE
workflow_comparison <-
  function(x, h, update_every_) {
    require(randtests)

    len <- length(x)
    frq <- frequency(x)
    ngroups <- len %/% update_every_

    split_x <- split(x, rep(1:ngroups, each = update_every_))

    results_by_block <- vector("list", ngroups)
    for (i in 18:(ngroups-h)) {## this needs to change if update_every_ != 1
      cat("Block",i,"/",ngroups-h,"\n")

      tr <- split_x[seq_len(i)]
      tr <- unname(do.call(c, tr))
      tst <- split_x[(i+1):(i+h)]
      tst <- unname(do.call(c, tst))

      ts_proc <-
        preprocessing_timeseries(tr = tr,
                                 tst = tst,
                                 frq = frq,
                                 k = 10)


      results <-
        apply_forecasting(train_adj = ts_proc$train_adj,
                          train_adj_xi = ts_proc$train_adj_xi,
                          train = ts_proc$train,
                          train_xi = ts_proc$train_xi,
                          embedded_tr = ts_proc$embedded_tr,
                          seasonal_factor = ts_proc$seasonal_factor,
                          lambda = ts_proc$lambda,
                          h = h)

      Y_hat <- results$Y_hat
      Y_hat <- do.call(rbind, Y_hat)
      Y_hat <- as.data.frame(Y_hat)

      raw_tst <- InvBoxCox(ts_proc$test, ts_proc$lambda)

      results_by_block[[i]] <-
        list(Y_hat = Y_hat,
             Time = results$Time,
             Test = raw_tst,
             TestBC = ts_proc$test,
```

```
            Train = tr)
    }
    results_by_block
  }

#MAIN
library(tsensembler)
library(forecast)
library(Metrics)

ts_len <- 500

# orizzonte previsivo
h <- 18

update_every_ <- 1

ngroups <- ts_len %/% update_every_
max_len <- ngroups * update_every_
len <- length(ts_list)

IDS <- 1:length(ts_list)

EXPRES <- vector("list",len)

for (i in IDS) {
  cat("TIME-SERIES",i,"/",len,"\n\n\n")
  x <- ts_list[[i]]
  x <- head(x, max_len)

  r_perf <- workflow_comparison(x = x,
                                h = h,
                                update_every_ = update_every_)

  EXPRES[[i]] <- r_perf
  save(EXPRES, file = "EXPRES.rdata")
}

# save(EXPRES, file = paste0("EXPRES_",IDS[1],"_",
#                            IDS[length(IDS)],".rdata"))
```

A.4 Analisi qualitativa

A.4.1 Funzioni utili

```
#grafico per le curve d'apprendimento
    plot_learning_curve <-
  function(x) {
    require(ggplot2)
    require(reshape2)

    x <- as.data.frame(x)

    #x$ID <- 18*(1:nrow(x))
    x$ID <- 18:(nrow(x)+18-1)

    df <- melt(x,id.vars = "ID")

    df$Model_Type <-
      !df$variable %in%
      c("arima","ets","tbats","naive","theta")

    df$Model_Type <-
      ifelse(df$Model_Type,
             "Apprendimento-Automatico",
             "Statistici")

    colnames(df) <- c("Data_Size","Model","AvgRank","Metodi")

    ggplot(df, mapping = aes(x=Data_Size,y=AvgRank)) +
      geom_smooth(aes(color=Metodi),lwd=2) +
      geom_line(aes(group=Model,color=Metodi),lwd=.5) +
      theme_minimal() +
      geom_vline(xintercept = 144) +
      theme(legend.position = "top") +
      xlab("Dimensione-dell'insieme-di-addestramento") +
      ylab("Rango-medio")
  }

avg_rank_plot <-
  function(avg, sdev) {
    require(reshape2)
    require(ggplot2)

    ord <- names(sort(avg))

    methods <- names(avg)

    ds <- data.frame(avg=avg,sdev=sdev, methods=methods, row.names = NULL)
```

```
    ds$methods <- factor(ds$methods, levels = ord)

    #ds <- melt(ds)

    ggplot(data = ds,
           aes(x = methods,
               y = avg)) +
      geom_bar(stat="identity",
               fill="#33CCCC") +
      theme_minimal() +
      theme(axis.text.x  = element_text(angle = 35,
                                        size = 12,
                                        hjust = 1)) +
      theme(axis.text.y  = element_text(size = 12),
            axis.title.y = element_text(size = 12)) +
      # geom_errorbar(aes(ymin = avg - sdev,
      #                   ymax = avg + sdev),
      #               width = .5,
      #               position = position_dodge(.9)) +
      labs(x="",
           y="Avg-Rank",
           title = "")
  }

#grafico per la complessita' computazionale
cc_plot <-
  function(x) {
    require(reshape2)
    require(ggplot2)

    avg_ <- colMeans(x)
    avg <- log(colMeans(x)+1)
    sdev <- log(apply(x,2,sd)+1)

    ord <- names(sort(avg))

    methods <- names(avg)

    ds <- data.frame(avg=avg,sdev=sdev, methods=methods, row.names = NULL)
    ds$methods <- factor(ds$methods, levels = ord)

    #ds <- melt(ds)

    ggplot(data = ds,
           aes(x = methods,
               y = avg)) +
      geom_bar(stat="identity",
               fill="#33CCCC") +
      geom_text(aes(label=round(avg_)),
                vjust=1.3,
                color="black",
                size=4)+
      theme_minimal() +
      theme(axis.text.x  = element_text(angle = 35,
                                        size = 12,
                                        hjust = 1)) +
      theme(axis.text.y  = element_text(size = 12),
            axis.title.y = element_text(size = 12)) +
      # geom_errorbar(aes(ymin = avg - sdev,
      #                   ymax = avg + sdev),
      #               width = .5,
      #               position = position_dodge(.9)) +
      labs(x="",
           y="CC-rispetto-al-Naive-stagionale",
           title = "")
  }

computational_complexity <-
  function(X) {
    cc <-
      sapply(X,
             function(x) {
               x <- x[!sapply(x,is.null)]

               cc_ <-
                 sapply(x,
                        function(y) {
                          y$Time
                        })

               rowSums(cc_)
             })

    cc <- as.data.frame(t(cc))
    cc[] <- lapply(cc, function(x) x / cc$naive)
    colnames(cc) <-
      c("ARIMA","ETS","Tbats","Theta",
        "Naive2","RF","GP","RBR","MARS","GLM")

    cc_plot(cc)
  }


eval_by_block_multi <-
  function(x) {
    x <- x[!sapply(x,is.null)]

    r <- lapply(x,
                function(o) {
```

```
                    tr <- o$Train
                    frq <- frequency(o$Test)

                    if (length(tr) > frq) {
                      frq_ <- frq
                    } else {
                      frq_ <- 1
                    }
                    tr <- ts(tr, frequency = frq_)

                    apply(o$Y_hat,1, function(yh) {
                      mean(mase_cal(insample = tr,
                                    outsample = as.vector(o$Test),
                                    forecasts = yh),na.rm = T)
                    })
                  })

    r <- do.call(rbind, r)

    r
  }

eval_by_block_single <-
  function(x) {
    x <- x[!sapply(x,is.null)]

    r <- lapply(x,
                function(o) {
                  tr <- o$Train
                  frq <- frequency(o$Test)

                  if (length(tr) > frq) {
                    frq_ <- frq
                  } else {
                    frq_ <- 1
                  }
                  tr <- ts(tr, frequency = frq_)

                  se_mdl <-
                    vapply(o$Y_hat[,1],
                           function(yh) {
                             mean(mase_cal(insample = tr,
                                           outsample = as.vector(o$Test[1]),
                                           forecasts = yh),na.rm = T)
                           }, double(1L))

                  names(se_mdl) <- rownames(o$Y_hat)

                  se_mdl
                })

    r <- do.call(rbind, r)

    r
  }
```

A.4.2 Grafici e tabelle

```
library(tsensembler)
#load("FINAL_EXPRES.rdata")
#EXPRES <- EXPRES_A
#rm(EXPRES_A)

# Calcolo mase
## h=18
resultsMultiStep <- lapply(EXPRES, eval_by_block_multi)
## h=1
resultsOneStep <- lapply(EXPRES, eval_by_block_single)

# calcolo rango h=1
ranksOneStep <-
  lapply(resultsOneStep,
         function(x) {
           t(apply(x,1,rank))
         })

# calcolo rango h=1, senza Naive stagionale
ranksOneStep_withoutNaive <-
  lapply(resultsOneStep,
         function(x) {
           t(apply(x[,-5],1,rank))
         })

# Compute rango h=18
ranksMultiStep <-
  lapply(resultsMultiStep,
         function(x) {
           t(apply(x,1,rank))
         })

# caloclo rango h=18, senza Naive stagionale
ranksMultiStep_withoutNaive <-
  lapply(resultsMultiStep,
```

```
          function(x) {
            t(apply(x[,-5],1,rank))
          })

# ranksOneStep <- resultsOneStep
# calcola la media dei ranghi per ogni punto dati nell'oggetto(ranksOneStep contiene i
#ranghi ottenuti per l'analisi a un passo)

avgRankOS <- apply(simplify2array(ranksOneStep), 1:2, mean, na.rm=TRUE)

# esegue la stessa operazione di avgRankOS, ma viene utilizzato l'oggetto ranksOneStep_withoutNaive che
#non include il modello "Naive" nei calcoli
avgRankOS_woNaive <- apply(simplify2array(ranksOneStep_withoutNaive), 1:2, mean, na.rm=TRUE)
#  calcola la mediana dei risultati originali (non i ranghi) per ogni punto dati nell'oggetto resultsOneStep
avgResOS <- apply(simplify2array(resultsOneStep), 1:2, median, na.rm=TRUE)

# data frame
avgRankOS <- as.data.frame(avgRankOS)
avgRankOS_woNaive <- as.data.frame(avgRankOS_woNaive)
avgResOS <- as.data.frame(avgResOS)

# STESSA COSA MA CON H = 18
avgRankMS <- apply(simplify2array(ranksMultiStep), 1:2, mean, na.rm=TRUE)
avgRankMS_woNaive <- apply(simplify2array(ranksMultiStep_withoutNaive), 1:2, mean, na.rm=TRUE)
avgResMS <- apply(simplify2array(resultsMultiStep), 1:2, median, na.rm=TRUE)
avgRankMS <- as.data.frame(avgRankMS)
avgRankMS_woNaive <- as.data.frame(avgRankMS_woNaive)
avgResMS <- as.data.frame(avgResMS)

# Smoothing con una media mobile di ordine 50, i risultati senza questa operazione di lisciamento sono
#difficili da interpretare velocemente ad occhio
avgRankOS_Sm <- roll_mean_matrix(avgRankOS, 50)
avgRankOS_woNaive_Sm <- roll_mean_matrix(avgRankOS_woNaive, 50)
avgResOS_Sm <- roll_mean_matrix(avgResOS, 50)
avgRankMS_Sm <- roll_mean_matrix(avgRankMS, 50)
avgRankMS_woNaive_Sm <- roll_mean_matrix(avgRankMS_woNaive, 50)
avgResMS_Sm <- roll_mean_matrix(avgResMS, 50)


# Plotting one step
## Avg rank by sample size
# plot_learning_curve(avgRankOS)
## smoothed avg rank by sample size
plot_learning_curve(avgRankOS_Sm)
## smoothed avg rank by sample size, w/o naive
plot_learning_curve(avgRankOS_woNaive_Sm)
## smoothed avg res by sample size
plot_learning_curve(avgResOS_Sm) +
  ylab("MASE")

# Plotting multi step
## Avg rank by sample size
# plot_learning_curve(avgRankMS)
## smoothed avg rank by sample size
plot_learning_curve(avgRankMS_Sm)
## smoothed avg rank by sample size, w/o naive
plot_learning_curve(avgRankMS_woNaive_Sm)
## smoothed avg res by sample size
plot_learning_curve(avgResMS_Sm) +
  ylab("MASE")


### CComplexity
computational_complexity(X = EXPRES)

### RISULTATI MEDI PER OGNI MDOELLO DEL RANGO MEDIO

avgr <- round(apply(avgRankOS_Sm,2,mean),2)
names(avgr) <-
  c("ARIMA","ETS","Tbats","Theta","Naive2",
    "RF","GP","RBR","MARS","GLM")
avgr <- sort(avgr)
avgrMS <- round(apply(avgRankMS_Sm,2,mean),2)
names(avgrMS) <-
  c("ARIMA","ETS","Tbats","Theta","Naive2",
    "RF","GP","RBR","MARS","GLM")

avgrMS <- sort(avgrMS)
```

Bibliografia e sitografia

Antal, D. (2022). *The dataset R Package: Create Data Frames that are Easier to Exchange and Reuse*. DOI: 10.5281/zenodo.7110256. URL: https://dataset.dataobservatory.eu/index.html.

Assimakopoulos, V. e K. Nikolopoulos (ott. 2000a). «The theta model: A decomposition approach to forecasting». In: *International Journal of Forecasting* 16, pp. 521–530. DOI: 10.1016/S0169-2070(00)00066-2.

— (2000b). «The theta model: a decomposition approach to forecasting». In: *International journal of forecasting* 16.4, pp. 521–530.

Azzalini, A. e B. Scarpa (2012). *Data analysis and data mining: An introduction*. OUP USA.

Box, G.E.P. e D.R. Cox (1964). «An analysis of transformations». In: *Journal of the Royal Statistical Society: Series B (Methodological)* 26.2, pp. 211–243.

Box, G.E.P. e G.M. Jenkins (1976). *Time Series Analysis: Forecasting and Control*. Holden-Day series in time series analysis and digital processing. Holden-Day. ISBN: 9780816211043. URL: https://books.google.it/books?id=1WVHAAAAMAAJ.

Brown e Robert G. (1957). «Exponential smoothing for predicting demand». In: *Operations Research*. Vol. 5. 1. Inst Operations Research Management Sciences 901 ELKRIDGE LANDING RD, STE ..., pp. 145–145.

Canova e B. Hansen (1995). «A Test for Seasonal Stability». In: *Journal of Business & Economic Statistics*.

Cerqueira, V., L. Torgo e C. Soares (2019). «Machine learning vs statistical methods for time series forecasting: Size matters». In: *arXiv preprint arXiv:1909.13316*.

Chan, Kung-Sik et al. (2022). «Package 'TSA'». In: *R package version* 1.

De Livera, A.M., R.J. Hyndman e R.D. Snyder (2011). «Forecasting time series with complex seasonal patterns using exponential smoothing». In: *Journal of the American statistical association* 106.496, pp. 1513–1527.

Friedman, J., T. Hastie e R. Tibshirani (2010). «Regularization paths for generalized linear models via coordinate descent». In: *Journal of statistical software* 33.1, p. 1.

Friedman, Jerome H (1991). «Multivariate adaptive regression splines». In: *The annals of statistics* 19.1, pp. 1–67.

Gardner Jr, Everette S. e McKenzie (1985). «Forecasting trends in time series». In: *Management science* 31.10, pp. 1237–1246.

Green, Peter J. e B.W. Silverman (1993). *Nonparametric regression and generalized linear models: a roughness penalty approach*. Crc Press.

Guerrero, V.M. (1993). «Time-series analysis supported by power transformations». In: *Journal of forecasting* 12.1, pp. 37–48.

Hastie, T., R. Tibshirani e J.H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

Hyndman, R.J. (2014). «with contributions from George Athanasopoulos». In: *Razbash, S., Schmidt, D., Zhou, Z., Khan, Y., Bergmeir, C., Wang, E.: forecast: Forecasting functions for time series and linear models*.

Hyndman, R.J. e G. Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, R.J., G. Athanasopoulos, C. Bergmeir et al. (2020). «Package 'forecast'». In: *Online] https://cran. r-project. org/web/packages/forecast/forecast. pdf*.

Hyndman, R.J., G. Athanasopoulos, S. GGally et al. (2022). «Package 'fpp2'». In.

Hyndman, R.J., A.B. Koehler et al. (2008). *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media.

Hyndman, Rob J e Yeasmin Khandakar (2008). «Automatic time series forecasting: the forecast package for R». In: *Journal of statistical software* 27, pp. 1–22.

Kang, Y., R.J. Hyndman e K. Smith-Miles (2017). «Visualising forecasting algorithm performance using time series instance spaces». In: *International Journal of Forecasting* 33.2, pp. 345–358.

Karatzoglou, A. et al. (2004). «kernlab-an S4 package for kernel methods in R». In: *Journal of statistical software* 11.9.

Kennel, M.B., R. Brown e H. Abarbanel (1992). «Determining embedding dimension for phase-space reconstruction using a geometrical construction». In: *Physical review A* 45.6, p. 3403.

Kuhn, M. et al. (2014). «Cubist: rule-and instance-based regression modeling». In: *R package version 0.0* 13.

Kwiatkowski, Denis et al. (1992). «Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?» In: *Journal of econometrics* 54.1-3, pp. 159–178.

Makridakis, Spyros e M. Hibon (2000). «The M3-Competition: results, conclusions and implications». In: *International journal of forecasting* 16.4, pp. 451–476.

Makridakis, Spyros, E. Spiliotis e V. Assimakopoulos (2018). «Statistical and Machine Learning forecasting methods: Concerns and ways forward». In: *PloS one* 13.3, e0194889.

Milborrow, S. et al. (2017). «earth: Multivariate adaptive regression splines». In: *R package version* 5.2.

Quinlan, J.R. et al. (1992). «Learning with continuous classes». In: *5th Australian joint conference on artificial intelligence*. Vol. 92. World Scientific, pp. 343–348.

Stoffer, D. e M.D. Stoffer (2023). «Package 'astsa'». In: *blood* 8, p. 1.

Takens, F. (1981). «Dynamical systems and turbulence». In: *Warwick, 1980*, pp. 366–381.

Wang, J. (2020). «An intuitive tutorial to Gaussian processes regression». In: *arXiv preprint arXiv:2009.10862*.

Wang, X., K. Smith e R.J. Hyndman (2006). ≪Characteristic-based clustering for time series data≫. In: *Data mining and knowledge Discovery* 13, pp. 335–364.

Wickham, H. (2011). ≪ggplot2≫. In: *Wiley interdisciplinary reviews: computational statistics* 3.2, pp. 180–185.

Williams, Christopher e C. Rasmussen (1995). ≪Gaussian processes for regression≫. In: *Advances in neural information processing systems* 8.

Wolpert, David H (1996). ≪The lack of a priori distinctions between learning algorithms≫. In: *Neural computation* 8.7, pp. 1341–1390.

Wright, M.N. e A. Ziegler (2015). ≪ranger: A fast implementation of random forests for high dimensional data in C++ and R≫. In: *arXiv preprint arXiv:1508.04409*.

Che il mio percorso posso fungere da esempio per i fratelli, non in quanto tale, bensì come manifestazione tangibile di piena realizzazione e rivalsa.

Ti giuro ora son contento,
adesso la sto prendendo
la corona d'alloro,
si frà dall'oro in oro grezzo
sulla testa del figlio,
si frà del figlio del cemento