

RINGRAZIAMENTI

Ringrazio tutti coloro che mi hanno aiutato a sviluppare la tesi rispondendo alle mie domande, in particolare ringrazio:

Prof. John R. Wolberg

Per aver risposto celermente e con costanza a tutte le mie domande su argomenti del suo libro (vedi Appendice e-mail);

Dr. Luigi Alfonso Ceron

Per avermi fatto conoscere la professione del *trader*, per avermi trasmesso le nozioni basilari sui mercati finanziari e l'analisi tecnica;

Prof. Cyril Goutte

Per avermi inviato alcuni *M-files* riguardanti modelli di regressione non parametrica;

Dr. Ronen Kimche

Per avermi inviato la versione demo *FKR*, programma progettato e sviluppato dal team guidato dal *Prof. John R. Wolberg*;

Prof. Louis Torgo

Per aver risposto ad alcuni miei dubbi riguardanti alcuni suoi *full paper*;

INDICE

1. INTRODUZIONE.....	4
1.1 TECNICHE DI MODELLAMENTO	5
1.2 MODELLARE I MERCATI FINANZIARI.....	7
1.3 LE SCUOLE DI PENSIERO SULLO STUDIO DEI MERCATI.....	11
1.4 I CANDIDATE PREDICTORS.....	14
1.5 IL MERCATO FINANZIARIO D'INTERESSE: IL FIB30	18
2. KERNEL REGRESSION	21
2.1 CONCETTI BASILARI.....	22
2.2 LA FUNZIONE KERNEL	26
2.3 LA <i>BANDWIDTH</i>	28
2.4 L'ORDINE DEL POLINOMIO	31
2.5 LA DIMENSIONALITÀ DEL POLINOMIO	34
2.6 MISURE DI VALUTAZIONE DEL MODELLO	39
3. KR AD ALTA PERFORMANCE.....	43
3.1 DALLE <i>BANDWIDTH</i> AL <i>P-TREE</i>	44
3.2 IL <i>P-TREE</i>	46
3.3 L'UTILIZZO DEL <i>P-TREE</i>	50
3.4 COMPLESSITÀ COMPUTAZIONALE	52
3.5 IL TEMPO PESA I DATI.....	58
3.6 DAY TRADING ED INTRADAY TRADING.....	60
4. STUDIO DI FATTIBILITÀ	61
4.1 PRESENTAZIONE DELLA MATRICE DATI	62
4.2 PRESENTAZIONE DEI PARAMETRI PRINCIPALI.....	64
4.3 DESCRIZIONE DEI RISULTATI OTTENUTI.....	66
4.4 UN APPROCCIO CON LE RETI NEURALI ARTIFICIALI.....	73
4.5 CONSIDERAZIONI FINALI.....	76
APPENDICE E-MAIL	79
BIBLIOGRAFIA	82

1. INTRODUZIONE

1.1 TECNICHE DI MODELLAMENTO

Il modellamento è un processo in cui si ricorre ai dati per determinare un modello matematico che può essere esplicitato mediante la seguente notazione:

$$Y = f(X),$$

dove Y rappresenta la variabile dipendente e X la variabile indipendente.

Esistono molte tecniche per modellare dati, in genere si distinguono due macro categorie: metodi parametrici e metodi non parametrici.

Sono detti parametrici quei metodi i cui modelli si reggono sull'ipotesi che $f(X)$ appartenga ad una famiglia di funzioni parametriche con un struttura fissa e con i parametri stimabili dai dati. Compito del processo di modellamento è tentare di trovare in modo empirico o in base a considerazioni teoriche i valori dei parametri di $f(X)$ più corretti e precisi e stabilirne la loro robustezza nel tempo con metodi d'inferenza statistica, oppure empiricamente utilizzando un *validation set*.

Alcuni esempi di modelli parametrici, fra i tanti che si potrebbero citare, sono i modelli *ARIMA* per i quali alcune considerazioni teoriche sono basate sullo studio delle correlazioni globali e parziali. Analizzando la forma dei due correlogrammi, infatti, si può restringere il campo di valori che possono assumere gli ordini del modello p e q , corrispondenti alla componente autoregressiva (*AR*) e media mobile (*MA*), diminuendo così le prove per la ricerca del modello migliore.

In alcuni problemi, però, si possono fare pochissime assunzioni concernenti la struttura di $f(X)$, si deve allora proporre una lista di potenziali pre-

dittori, i quali, con pesi diversi, si ritiene possano concorrere alla stima di $f(X)$. In questo caso l'approccio più utilizzato consiste nello sviluppare modelli basati sui dati senza aver la pretesa di trovare una struttura fissa e parametrizzata per il modellamento. Queste tecniche sono dette non parametriche o *data-driven methods*.

L'applicazione dei modelli non parametrici richiede l'uso intenso del computer a causa dell'enorme quantità di calcoli che devono essere effettuati; l'attenzione perciò deve essere posta nella ricerca di algoritmi molto efficienti, che apprendano dai dati e che riescano a sviluppare modelli multidimensionali con un certo grado di potere previsionale e con un tempo computazionale ragionevole.

Si può altresì combinare tecniche di modellamento parametrico con quelle non parametriche. Una possibile strategia è di ricorrere a metodi non parametrici per esplorare i principi che sembrano guidare i dati, e in un secondo momento, studiare i risultati ottenuti per specificare un modello parametrico. Tuttavia, più la dimensionalità del modello e la non linearità dei dati crescono, sempre più remota diventa la speranza di trovare un modello parametrico: è per questo motivo che nel campo finanziario i metodi non parametrici hanno riscontrato un notevole interesse.

All'interno della grande classe dei metodi non parametrici si distinguono due importanti metodologie: le reti neurali e le regressioni non parametriche.

1.2 MODELLARE I MERCATI FINANZIARI

Nella maggior parte dei mercati finanziari le contrattazioni avvengono in continuo, concentrate in un periodo compreso tra l'apertura della giornata borsistica e la chiusura della stessa. Per sintetizzare le quotazioni, espresse in punti, in percentuale oppure in prezzi, in un dato intervallo di tempo, si ricorre a quattro valori: il valore di apertura (*Open*), il valore di chiusura (*Close*), il valore minimo (*Low*), il valore massimo (*High*) ed inoltre si indica il numero di contratti scambiati (*Volume*) e spesso anche il numero di contratti aperti (*Open-Interest*). Perciò le quotazioni storiche saranno espresse da perlomeno cinque serie storiche anche se a volte viene presa in considerazione solo quella dei valori di chiusura.

Una serie storica è una successione ordinata di numeri reali che misura un certo fenomeno seguendo un preciso ordine temporale. Lo studio di tale successione trova la propria ragion d'essere nel fatto che i valori passati di una serie possono influenzare i valori futuri, i quali possono dipendere dalla presenza di alcune forze deterministiche sottostanti che possono influenzare sia la media (il *trend*), sia la varianza (la volatilità). Queste componenti e le relazioni non lineari che intercorrono tra i dati dovranno essere appresi e riconosciuti dal processo di modellamento. Quanto meglio il metodo riesce ad apprendere e quanto più i dati dipendono dalle forze sottostanti, tanto migliori saranno le previsioni sulle quali fondare un sistema automatizzato o *trading system*.

I *trading system* sono sistemi progettati *ad hoc* per dare al *trader* delle direttive sulle operazioni da effettuare. Generalmente queste direttive sono date da tre tipi di segnali: compra (*buy*), vendi (*sell*) e tieni (*hold*).

Ogni *trading system* dovrebbe essere progettato in modo diverso in base ai seguenti fattori:

- il tipo di mercato cui appartiene la serie storica oggetto di modellamento (azionario, a reddito fisso, dei derivati);
- la frequenza di *trading* che si intende perseguire (*intraday*, *daily*, *weekly*);
- la frequenza di campionamento, ovvero l'unità di grandezza della variabile tempo utilizzata: *Tick*, 1', 5', 60', un giorno, ecc...

Per modellare i mercati finanziari si devono tener presente le seguenti osservazioni:

- non si conosce a priori quali siano le variabili indipendenti necessarie per realizzare un buon modello;
- non si sa a priori quante variabili indipendenti proporre per essere sicuri di trovare un modello con un certa capacità previsionale;
- non è scontato che si riesca ad ottenere un buon modello con i dati a disposizione;
- per ogni combinazione delle variabili indipendenti esiste un *range* di possibili valori della variabile dipendente: non è detto che per uguali valori delle variabili indipendenti in due punti, il valore della variabile dipendente sia lo stesso;
- è impossibile ottenere un modello privo della componente d'errore, l'obiettivo è costruire un modello in cui il segnale è forte abbastanza da permettere di fare delle buone previsioni.

I mercati finanziari sono di solito caratterizzati da una bassa frazione di segnale rispetto a quella di rumore (*low signal to noise ratio*), in altre parole, un forte cambiamento di prezzo da un periodo al successivo sembra essere casuale (*random shock*), per di più la bassa componente di segnale tipica-

mente varia in modo fortemente non lineare. Ciononostante l'andamento dei prezzi non apparirà del tutto casuale se verranno considerate le variabili indipendenti o *candidate predictors*: se vengono ricavate delle informazioni, se pur povere, da ciascuna serie storica proposta come *predictor*, l'assemblaggio di tutte le informazioni ottenute può portare a spiegare gran parte del cambiamento dei prezzi.

L'analista deve porre, dunque, molta attenzione nella scelta dei *candidate predictors* che dovranno essere in ritardo (*backward looking*) rispetto alla serie da prevedere, in altre parole, i valori dei *predictors* devono essere conosciuti nel momento in cui si vogliono fare le previsioni, in modo tale che possano dare informazioni sul futuro (vedi Appendice e-mail).

Dopo aver presentato il set dei potenziali predittori, questi devono essere valutati, prima singolarmente e poi congiuntamente per trovare i *best predictors* al fine di ridurre la dimensione e la complessità dei modelli da considerare nella valutazione finale.

È qui che tecniche regressive non parametriche, come la *kernel regression*, danno i migliori risultati in termini di efficacia ed efficienza a confronto con altre metodologie quantitative quali le reti neurali, che nonostante tutto rimangono forse più idonee nelle previsioni, una volta ridotto il numero di potenziali predittori con la *kernel regression* o con “motori di ricerca” appartenenti ad altre metodologie.

Il problema, infatti, dell'utilizzo delle reti neurali per modellare i mercati finanziari è dato dall'enorme quantità di tempo computazionale necessario per generare il modello, perciò per avere qualche speranza di successo è indispensabile pre-processare i dati per cercare di ridurre il numero di *candidate predictors* ad una quantità compatibile con la potenza del computer di cui si dispone e il tempo a disposizione.

Riepilogando, il processo di modellamento dei mercati finanziari può essere schematizzato come segue:

1. specificare una lista di *candidate predictors* (matrice X) e riunire i dati appropriati per rilevare ogni vettore di X nel periodo preposto per il modellamento (cap. 1.4);
2. per lo stesso periodo aggiungere i valori passati della variabile dipendente Y , la serie storica che dovrà essere modellata;
3. determinare la dimensionalità massima (d_{max}) del modello (cap. 2.5);
4. specificare un criterio per valutare ogni spazio del modello ossia ogni *candidate predictors* e loro combinazioni (cap. 2.6);
5. specificare una strategia per esplorare gli spazi (cap. 2.5, 3.2, 3.3);
6. valutare ogni spazio con il criterio stabilito (punto 4) per determinare i migliori modelli;
7. se ci sono dati sufficienti, testare i migliori modelli trovati impiegando dei dati che non sono stati inseriti nel processo di modellamento, gli *out-of-sample points* o *evaluation set* (cap. 2.1).

1.3 LE SCUOLE DI PENSIERO SULLO STUDIO DEI MERCATI

Lo studio dei mercati finanziari ha portato alla crescita e allo sviluppo di tre diverse scuole di pensiero: l'analisi fondamentale, la teoria *random walk*, l'analisi tecnica.

L'analisi fondamentale

L'analisi fondamentale è basata su statistiche economiche e finanziarie, mira a determinare il valore intrinseco di società quotate al fine di individuare quelle che appaiono sottovalutate rispetto ai prezzi espressi dal mercato.

Con questo tipo di analisi si cerca, dunque, di determinare lo scostamento tra il prezzo attuale di mercato e il prezzo stimato attualizzando i flussi di reddito futuri che la società quotata potrà generare ipotizzando un ambiente operativo perfetto.

Questa metodologia presenta però delle difficoltà operative:

- l'irrazionalità del comportamento dei partecipanti al mercato;
- il problema della stima dei futuri flussi di reddito relativi al titolo analizzato;
- il tempo di aggiornamento, di solito trimestrale, che rende impossibile un'operatività frequente come per esempio quella *intraday*.

La teoria *random walk*

La teoria *random walk* si fonda sull'ipotesi di efficienza dei mercati finanziari: secondo tale teoria i prezzi di un qualsiasi bene quotato fluttuano irregolarmente e casualmente attorno al proprio valore intrinseco, spinti dalle informazioni che continuamente arrivano sul mercato. Poiché tali informazioni arrivano in maniera imprevedibile, non è possibile prevedere il futuro

andamento dei prezzi, di conseguenza, la miglior strategia di mercato è del tipo *buy and hold*.

I limiti di questa teoria consistono nel porre come premessa teorica il fatto che tutti gli operatori abbiano pari accesso alle informazioni e che queste ultime siano effettivamente conosciute e analogamente valutate da tutti i partecipanti al mercato.

L'analisi tecnica

L'analisi tecnica è lo studio del comportamento del mercato condotto attraverso l'esame, non solo dal punto di vista grafico ma anche con l'utilizzo di opportuni indicatori, delle serie storiche dei prezzi e dei volumi di scambio.

Si basa sostanzialmente su tre premesse basilari:

- il mercato sconta tutto: nei prezzi sono già incorporati tutti quei fattori di tipo fondamentale, politico, psicologico che ne hanno determinato l'andamento, con la conseguenza che ogni informazione disponibile è già riflessa nel prezzo;
- i prezzi si muovono per tendenze: trend rialzisti, ribassisti e laterali;
- la storia si ripete: il movimento dei prezzi e dei volumi, essendo il risultato di una somma di azioni umane, riflette, attraverso andamenti ricorrenti e relativamente uniformi, la psicologia e il comportamento dell'uomo. Per comprendere il futuro è quindi essenziale studiare il passato, poiché il futuro potrebbe esserne una ripetizione o presentare forti analogie con la storia.

La presa di posizione dell'analista su una di queste tre teorie ne influenza certamente il modo di operare sui mercati finanziari e il criterio di selezionare i potenziali predittori da inserire nel processo di modellamento al fine di

creare un *trading system*. In particolare, un investitore che adotta la teoria del *random walk* non avrà mai l'interesse di costruire modelli quantitativi e di cercare dei predittori e preferirà agire come un "cassettista". Di conseguenza l'insieme dei candidate predictors sarà composto da indicatori statistici e matematici formulati dall'analisi fondamentale e l'analisi tecnica.

1.4 I CANDIDATE PREDICTORS

Una delle prime cose da fare per modellare i mercati finanziari è quella di proporre un set di *candidate predictors*.

Se si sapesse cosa guida il mercato i *candidate predictors* non sarebbero necessari, in realtà però, non si conoscono, né le forze deterministiche sottostanti, né il modo in cui tali forze guidino il mercato.

Alcuni *candidate predictors* facili da calcolare sono le differenze, che in analisi tecnica vengono chiamate *momentum*. Questi indicatori servono per calcolare le variazioni del prezzo corrente rispetto al prezzo di periodi precedenti e consente di stimare la velocità con la quale i prezzi stanno aumentando o diminuendo.

Un altro tipo di *candidate predictor* sono le medie mobili (*MM*) che possono essere di vario tipo, le più diffuse sono le medie mobili semplici (MM^s), ponderate (MM^p) ed esponenziali (MM^e).

Le medie mobili utilizzate singolarmente non sono dei buoni predittori a causa del ritardo con cui danno i segnali: più il periodo delle medie mobili (k) cresce, più il ritardo nella previsione del cambiamento di direzione dei prezzi si fa maggiore; utilizzate, invece, congiuntamente con altre serie possono essere efficaci.

Se i *candidate predictors* non sono stazionari in media, allora vi è una grande probabilità che i valori osservati non siano sullo stesso *range* dei valori passati e il loro utilizzo per la previsione si rivelerebbe inappropriato, si deve ricorrere, perciò, a delle opportune trasformazioni. Ad esempio, la media mobile di una serie non stazionaria in media è anch'essa una serie non stazionaria, ma la frazione della serie con la sua media mobile da origine ad una

serie stazionaria con media pari a uno. In analisi tecnica questo oscillatore viene chiamato *pista ciclica*.

Non è un caso che molte trasformazioni di *candidate predictors* elementari costruiti con differenze e medie mobili diano vita ai principali indicatori dell'analisi tecnica.

Esempi di indicatori basati sulle differenze sono:

- il *Momentum*, accennato precedentemente, ed espresso in formula come segue:

$$M_k = P_t - P_{t-k} \text{ oppure } M_k = 100 * (P_t / P_{t-k});$$

- il *ROC (price Rate Of Change)*: rileva lo scostamento percentuale tra il prezzo corrente e il prezzo di una certa epoca precedente, consentendo di evidenziare la forza intrinseca dei movimenti di mercato. La formula per il calcolo del *ROC* è la seguente:

$$ROC = 100 * \left(\frac{P_t - P_{t-k}}{P_{t-k}} \right).$$

Alcuni esempi di indicatori basati invece sulle medie mobili sono:

- la *Pista Ciclica*: consente di esprimere la crescita o la diminuzione del prezzo depurato dalla propria tendenza e si calcola come segue:

$$PC = P_t / MM_k (P_t).$$

- le *Bande di Bollinger*. Sono due linee poste al di sopra e al di sotto di una media mobile dei prezzi secondo il valore della deviazione standard della media mobile stessa:

$$BB_{-}^{+} = MM_k (P_t) \pm \delta_{MM_k},$$

poiché la deviazione standard misura la variabilità media delle quotazioni, l'ampiezza della banda tenderà ad aumentare durante le fasi di mercato molto volatili e a contrarsi in caso di variazioni contenute dei prezzi;

- il *MACD* (*Moving Average Convergence-Divergence*): il rapporto tra due medie mobili esponenziali aventi differente ampiezza temporale:

$$MACD = \frac{MM_k^E}{MM_h^E};$$

- il *Relative Strength Index* (*RSI*):

$$RSI(k) = 100 - \left(\frac{100}{1 + MM_k(C^+) / MM_k(C^-)} \right),$$

dove $MM_k(C^+)$ e $MM_k(C^-)$ sono le medie mobili con ritardo pari a k delle serie storiche rispettivamente delle chiusure al rialzo (C^+) e delle chiusure al ribasso (C^-). L' *RSI* misura l'intensità direzionale dei movimenti di mercato consentendo di individuare situazioni di mercato ipercomprato (caratterizzate cioè da un eccesso di domanda) e situazioni di mercato ipervenduto (caratterizzate cioè da un eccesso di offerta).

Vi sono, inoltre, altri indicatori basati sia su differenze sia su medie mobili:

- il *Price Oscillator*: rileva le differenze tra due medie mobili di diverso dominio (k, h), cioè:

$$PO = MM_k - MM_h, \text{ con } k < h;$$

- il *Qstick*: si tratta di una media mobile semplice della differenza tra prezzo di chiusura e quello di apertura:

$$Q = MM_k^s(C - O);$$

- l'*Oscillatore Stocastico*, composto da due linee, la *signal line* (*%K*) e la *trigger line* (*%D*), calcolate sulla base di diversi intervalli temporali: consente di misurare la posizione relativa del prezzo di chiusura (*C*). La sua costruzione è basata sull'assunzione che, in una fase di mercato al rialzo, il valore delle chiusure giornaliere tende a collocarsi in prossimità dei prezzi massimi (*H*), mentre in una fase di mercato orientata al ribasso i prezzi di chiusura tendono ad avvicinarsi ai livelli minimi (*L*).

Le formule delle due linee sono:

$$\%K = 100 * \left(\frac{C - L_5}{H_5 - L_5} \right)$$

e

$$\%D = MM_3(\%K),$$

dove:

C = ultimo prezzo di chiusura,

*L*₅ = prezzo minimo degli ultimi cinque giorni,

*H*₅ = prezzo massimo degli ultimi cinque giorni.

1.5 IL MERCATO FINANZIARIO D'INTERESSE: IL FIB30

Come si è detto, un *trading system* deve essere costruito su misura in base al mercato nel quale si intende operare. Per elaborare un buon *trading system* ed effettuare degli scambi profittevoli è necessario, perciò, conoscere le principali caratteristiche e le regole del mercato d'interesse che in questo caso è il *Fib30*.

Il *Fib30* è un contratto di borsa negoziato sull'*IDEM* il cui oggetto sono *futures* aventi come *sottostante* l'indice *Mib30* elaborato dalla Borsa valori di Milano.

I *futures* sono particolari contratti standard posti in essere quando le parti concordano subito la quantità ed il prezzo, ma stabiliscono di differirne l'esecuzione ad una data futura. Un'operazione di questo tipo viene definita "a termine".

I *futures* sono scambiati sui mercati regolamentati ed i loro prezzi sono buoni indicatori delle prospettive sui mercati per consegna immediata. In particolare sono contratti di tipo standardizzato, le cui caratteristiche sono definite dalle autorità di mercato nel senso che gli investitori non possono determinare autonomamente le caratteristiche del contratto, ma devono selezionare fra i contratti scambiati sul mercato quello più idoneo a soddisfare le loro esigenze.

I contratti *futures* tutelano le parti dal rischio di insolvenza in quanto passano attraverso la negoziazione della *Cassa di compensazione e garanzia* che si pone come controparte sia del venditore sia del compratore, garantendo il buon fine delle operazioni tramite la riscossione del *margin* di *garanzia*, versamento da parte dell'intermediario al fondo costituito presso la *Cas-*

sa di compensazione e garanzia, che viene utilizzato in caso di inadempienza di un intermediario.

Il prezzo pattuito al momento della stipulazione del contratto rimane invariato per tutta la durata del rapporto; non si tiene conto, quindi, delle variazioni intervenute nel prezzo del bene dal momento dell'assunzione dell'impegno a quello della sua effettiva esecuzione, variazione che può essere al ribasso o al rialzo. Chiaramente otterrà un risultato positivo solo chi avrà previsto correttamente l'andamento dei prezzi.

Mentre in un contratto *futures* ordinario le parti si impegnano a scambiarsi, alla scadenza dei termini concordati, un certo quantitativo di titoli contro un prezzo prestabilito, nel caso di *futures* su indici il controvalore dei titoli viene determinato sulla base dell'indice di riferimento come prodotto tra il valore assegnato convenzionalmente a ciascun punto dell'indice e la differenza tra il valore dell'indice al momento della stipula del contratto e il valore che esso raggiungerà il giorno della scadenza. Nel *Fib30* l'indice di riferimento, sulle cui variazioni l'operatore di borsa intende speculare, è l'indice *Mib30* che esprime l'andamento dei trenta titoli più importanti quotati alla Borsa di Milano (*blue chips*) selezionati in base al grado di liquidità ed al livello di capitalizzazione.

Nel caso del *Fib30* il valore assegnato a ciascun punto dell'indice è pari a 5 euro, il prezzo del contratto sarà perciò dato dal prodotto tra il punteggio dell'indice e 5 euro. L'operatore finanziario è tenuto, però, a versare solo il 7,5% dell'importo così determinato (*c.d. margine iniziale*, cioè la percentuale minima del margine di garanzia), permettendogli, in tal modo, con un importo abbastanza contenuto, di operare su un capitale decisamente più consistente.

Il margine di garanzia deve essere sempre mantenuto costante, o meglio sopra il 7,5%, il che comporta per l'investitore l'obbligo di versamenti o prelevamenti (*c.d. margini di variazione giornalieri*), calcolati in base alle per-

dite o agli utili che registrerà l'indice di riferimento secondo il principio del *marking market*.

Le contrattazioni in *Fib30* sono possibili fino al terzo venerdì dei mesi in cui è fissata la scadenza (marzo, giugno, settembre e dicembre). Il primo giorno di Borsa aperta successivo alla scadenza si provvederà alla liquidazione dei contratti *Fib30*, che avviene in contanti (*cash settlement*) in quanto non è prevista la consegna fisica delle singole azioni che compongono il *Mib30*.

Il mercato è operativo dal 28 novembre 1994, la serie storica è dunque molto ricca di dati e non sarà necessario considerare il problema della scarsità dei dati necessari per il modellamento.

Il *MiniFib30* è anch'esso un contratto *futures* negoziato sull'*IDEM* ed è stato introdotto dalle modifiche del 19 Aprile 2000 al regolamento di borsa. Esso riproduce, in dimensioni più ridotte, il *Fib30* in quanto il valore assegnato a ciascun punto dell'indice *MiniFib30* è pari a 1 euro, invece che 5 come nel caso del *Fib30*.

Il valore non elevato dei punti indice combinato con gli effetti della leva finanziaria, propri di ogni strumento derivato, fanno sì che per operare in *MiniFib30* sia necessaria una liquidità minima contenuta (spesso inferiore a 5000 euro), rendendo tale strumento finanziario accessibile anche e soprattutto al piccolo investitore individuale. Tale contratto è soggetto a disposizioni regolamentari analoghe a quelle previste per il *Fib30* in materia di quotazione, modalità di negoziazione, scadenze e liquidazione.

2. KERNEL REGRESSION

2.1 CONCETTI BASILARI

La *kernel regression* è un metodo statistico non parametrico che appartiene al campo di ricerca usualmente chiamato modellamento di smussamento locale. È un metodo che non assume nessun tipo di generalizzazione sulla struttura dei dati fino al tempo in cui vengono effettuate le previsioni.

La *kernel regression* fonda la sua efficacia e la sua efficienza su varie caratteristiche:

- mancanza di qualsiasi tipo di ipotesi sui dati, se non quella di stazionarietà in media ed il più possibile in varianza;
- capacità di trattare efficientemente serie lunghissime di dati, anche decine di migliaia;
- capacità di sviluppare modelli con molte dimensioni;
- una buona componente tecnologica: la complessità computazionale gioca un ruolo rilevante nella scelta dei vari parametri che entrano in gioco nell'elaborazione del modello (cap. 3.4);
- può essere implementata con altre metodologie per generare algoritmi ibridi ad alta performance (cap. 4.5).

Tutte queste caratteristiche rendono la *kernel regression* un'applicazione interessante nel campo finanziario, come del resto in tutte quelle discipline in cui la mole di dati e la dimensione del modello è elevata.

Il primo passo da compiere per sviluppare la metodologia è stabilire l'insieme dei *candidate predictors*, ovvero l'insieme delle variabili indipendenti che si ritiene possano concorrere con pesi differenti all'apprendimento

e alla successiva previsione della variabile dipendente (y), tipicamente la variazione del prezzo di chiusura.

Normalmente quando si modellano dati finanziari il numero di *candidate predictors* è molto elevato, anche nell'ordine di centinaia, tuttavia questo numero è vincolato da:

- complessità del problema che si vuole analizzare (si veda cap. 2.5);
- risorse tecnologiche (si veda cap. 3.4);
- numero di *records* disponibili;

inoltre, il numero e la scelta dei *candidate predictors* dipende molto dalla bravura e dall'esperienza dell'analista: una cattiva selezione dei *candidate predictors*, infatti, porterebbe inevitabilmente a delle previsioni mediocri, se non pessime, anche se sviluppate con metodologie ottime.

Quando non si hanno abbastanza *records* nella serie storica oggetto d'analisi esistono vari modi di procedere:

- combinare le variabili in modo tale che una nuova variabile includa gli effetti di due o più variabili originali (es. analisi in componenti principali);
- usare una strategia di modellamento *multistage* (gli *output* di uno stadio sono gli *input* dello stadio successivo);
- diminuire l'ampiezza di campionamento dei dati, per esempio passare da intervalli di un'ora ad intervalli di 15 minuti, oppure da intervalli di 15' ad intervalli di 5' e così via fino ad arrivare all'intervallo più piccolo possibile chiamato *tick*;
- aumentare il numero di *records* combinando *set* di dati simili (*merging*).

Quando si usano dati per creare modelli è molto utile separarli in tre sottoinsiemi:

- *learning set (LRN)*: comprende la gran parte dei punti della serie di dati (~50-60%) e serve per far apprendere al modello l'andamento della serie storica per poi elaborare le previsioni;
- *testing set (TST)*: sono i punti che il modello deve in un primo momento stimare ed in un secondo confrontare con y per calcolare alcune misure di *performance* (cap. 2.6). Il *testing set* è determinante per scegliere i valori dei parametri e le combinazioni di *candidate predictors* che danno i risultati migliori in termini di misure di performance: con tecniche di tipo non parametrico, è un set indispensabile per la ricerca del modello;
- *evaluation set (EVL)*: sono punti delle serie storiche che sono tenuti da parte (*out-of-sample points*) per effettuare i test finali (*out-of-sample testing*). Questo set viene usato per valutare l'effettiva efficacia previsionale e la robustezza del modello.

Il *learning set* a sua volta può essere statico o dinamico. Si dice che il *learning set* è dinamico quando ogni *record* del *test set* che viene testato è aggiunto al *learning set*, cioè è utilizzato per aggiornare il *learning set* per stimare il *record* di test successivo.

Ci sono due modi di rendere il *learning set* dinamico:

- *growing option*: si lascia che il *learning set* aumenti man mano che ogni punto di test viene aggiunto;
- *moving window option*: per ogni *record* del *test set* aggiunto viene scartato il punto iniziale del *learning set*, mantenendo costante ad ogni iterazione il numero di punti nel *learning set*.

Il *learning set* si definisce statico (*static option*) quando, invece, rimane invariato in tutte le fasi del modellamento.

Riassumendo, le opzioni che saranno presenti nell'algoritmo per lo sviluppo della *kernel regression* per quanto riguarda l'aggiornamento del *learning set* sono tre: “*growing*”, “*moving window*”, “*static*”. È chiaro che i risultati che si possono ottenere con ciascuna delle tre opzioni sono differenti perché diversi sono i coefficienti dei polinomi che generano \hat{y}_j .

La scomposizione del data set può avvenire secondo tre differenti modalità:

- *null*: vengono considerate tutte le osservazioni del data set sia per *LRN*, che per *TST* e *EVL*;
- *sequential*: si ripartiscono in modo sequenziale le osservazioni del data set sulla base delle percentuali di ripartizione assegnate;
- *fractionary*: i *records* vengono ripartiti in modo “distribuito” sulla base delle percentuali di ripartizione assegnate. La differenza rispetto alla metodologia sequenziale è che i dati non sono successivi fra loro, ma sono distribuiti nell'intero data set.

Nello studio della *kernel regression* si è applicata la metodologia *sequential*, ritenuta più idonea, almeno nel *learning set*, per il modellamento di serie storiche ai fini predittivi; mentre, le percentuali di ripartizione si aggirano attorno al 50% per *LRN*, 25% per *TST* e *EVL* (cap. 4.4), oppure 50% nel *LRN* e 50% in *TST* (cap. 4.1, 4.3). Il *learning set* è sempre stato mantenuto statico, optando per l'opzione *static*.

Il numero totale dei punti di *learning*, di *testing* e di *evaluation* sarà identificato dai parametri *nln*, *ntst* e *nevl*.

2.2 LA FUNZIONE KERNEL

L'utilizzo della *kernel regression* prevede la scelta della funzione *kernel* che andrà a determinare i pesi applicati all' *i-esimo* punto di *learning* (y_i) per stimare il *j-esimo* punto di testing (y_j):

$$W(X_{i,d}, X_{j,d}, k), \quad (\text{eq. 2.2.1})$$

dove

$X_{i,d}$ è l'*i-esimo* valore di *LRN* del *d-esimo* candidate predictors;

$X_{j,d}$ è il *j-esimo* valore di *TST* della *d-esima* dimensione;

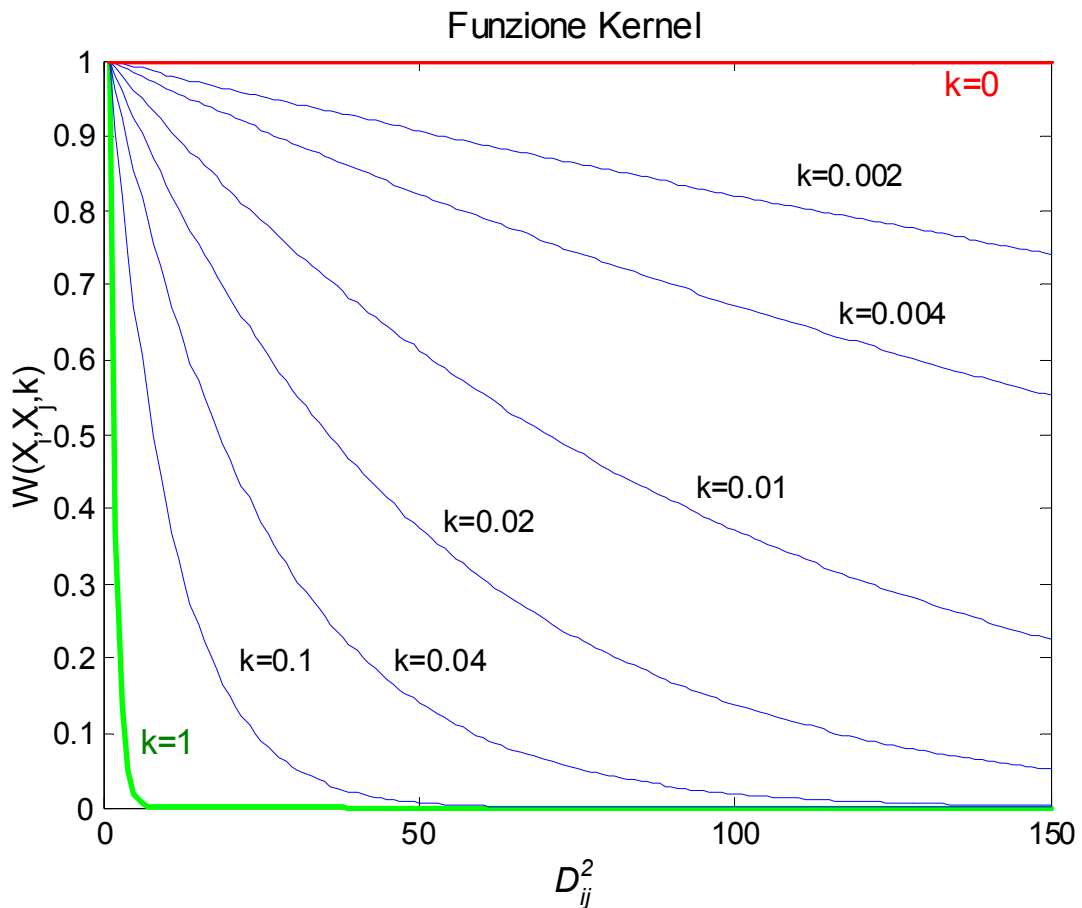
k è il parametro di smussamento del quale si dirà più avanti.

Ci sono tanti tipi di funzioni *kernel*, molti sono descritti da *W.Hardle* nel suo libro (vedi bibliografia) e nel *help* del pacchetto statistico *XploRe* da lui ideato, tuttavia in questo lavoro viene considerata solamente una funzione esponenziale (vedi Appendice e-mail):

$$W(X_{i,d}, X_{j,d}, k) = e^{-kD_{i,j}^2}, \quad (\text{eq. 2.2.2})$$

la cui forma cambia al variare del parametro k chiamato *smoothing parameter* e $D_{i,j}^2$, il valore della distanza al quadrato tra i punti delle variabili indipendenti del *learning set* e quelli del *test set*. Il parametro k può variare tra 0 e 1: se è pari a 0 tutti i punti sono ugualmente pesati, al suo crescere, invece, ai punti più vicini viene assegnato un peso maggiore rispetto a quelli più distanti.

Per mostrare come la funzione *kernel* agisce al variare di k e di $D_{i,j}^2$ si illustra di seguito il grafico che ha in ascissa la distanza al quadrato, e in ordinata il valore assunto da $W(X_i, X_j, k)$.



Si deduce chiaramente che più la distanza tra i punti di *LRN* e ciascun punto di *TST* è elevata, meno agisce il *kernel* e la ponderazione decresce esponenzialmente. Inoltre, più k è elevato più il *kernel* riduce l'effetto del peso sui punti distanti. Si noti che il compito del *kernel* non è quella di aumentare l'importanza dei punti più vicini, ma di diminuire l'influenza dei punti più lontani relativamente al parametro k .

2.3 LA BANDWIDTH

Il concetto di banda (*bandwidth*) completa la funzione *kernel* (eq. 2.2.2) ridefinendo l'equazione 2.2.1 come segue:

$$W(X_{i,d}, X_{j,d}, k, h), \quad (\text{eq. 2.3.1})$$

dove il nuovo parametro h stabilisce l'ampiezza dell'intervallo di banda.

La *bandwidth* è definita come la regione in cui il peso (W) esercita la sua influenza, in altre parole, per un punto x_j del *test set* solo i punti x_i del *learning set* più vicini a x_j ed interni all'intervallo $[x_j - h/2; x_j + h/2]$ sono utilizzati e pesati per stimare y .

Paradossalmente, se non ci sono punti di *learning* che cadono in questa regione, y non può essere stimato. Come soluzione a questo problema spesso si ricorre all'approccio *K-nearest neighbor estimate*, metodo che va a pesare solo i primi K punti più vicini al punto considerato. Comunque in presenza di un'alta densità di dati, l'uso di una *bandwidth* costante resta ancora una scelta ragionevole.

Per estendere il concetto della banda in p dimensioni è necessario ridefinire la distanza al quadrato della funzione *kernel* (eq. 2.2.2) e basare la *bandwidth* su questa distanza. È chiaro che, se le variabili indipendenti (X_d) avessero scale molto differenti, la dimensione più lunga dominerebbe totalmente il calcolo. Per risolvere questo problema bisogna utilizzare una sorta di distanza normalizzata.

Il metodo più comune per normalizzare una distanza è quello di dividere i valori di tutte le dimensioni per il relativo *range* ($\text{range} = x_{\max_d} - x_{\min_d}$) oppure per la *deviazione standard* corrispondente ad ogni dimensione.

In uno spazio a più dimensioni il valore della distanza normalizzata tra il j -esimo punto del *test set* e l' i -esimo punto di *learning* è calcolata come segue:

$$D_{i,j}^2 = \sum_{d=1}^p \left(\frac{x_{i,d} - x_{j,d}}{N_d} \right)^2, \quad (\text{Eq. 2.3.2})$$

dove $x_{i,d}$ è la variabile della d -esima dimensione nell' i -esimo punto di *learning*, e allo stesso modo, $x_{j,d}$ è la variabile della d -esima dimensione per il j -esimo punto di *testing*; N_d è invece la costante di normalizzazione (*range* o *deviazione standard*) per la d -esima dimensione.

Solitamente è più vantaggioso lavorare con $D_{i,j}^2$ piuttosto che con $D_{i,j}$ perché si elimina la necessità di calcolare le radici quadrate, risparmiando tempo computazionale.

La ragione primaria dell'introduzione del concetto di banda è quella di eliminare l'effetto dei punti molto distanti nella stima di un dato punto di *testing*. A tal fine si potrebbe pensare che una costante di smussamento (k) molto elevata (~ 1) possa risolvere il problema, ma così facendo si rischierebbe di ridurre troppo lo smussamento fino ad arrivare al caso limite di ottenere una stima basata primariamente solo sul singolo punto più vicino ad ogni punto di test.

Il secondo motivo per il quale converrebbe l'uso della banda h oppure del *K-nearest neighbor estimate*, è dato dal risparmio in tempo di calcolo: la complessità computazionale rimane $O(n \ln n * ntst)$ per quanto concerne il calcolo delle distanze, ma ricorrendo alla banda si riducono ad h i punti di *learning* pesati per stimare ogni valore di *test*.

Se si pensa che, di norma, si possono avere valori di $n \ln n$ e $ntst$ nell'ordine di alcune migliaia e che già nell'inizializzazione di una matrice di

tali dimensioni certe applicazioni e/o computer si bloccano, tale riduzione dei calcoli (h è nell'ordine delle centinaia) non è un risultato da poco.

Tuttavia, per diminuire ulteriormente la complessità computazionale, che rimane pur sempre elevata, si utilizzano sistemi di regressione *kernel* ad alta *performance* con l'inserimento del *p-Tree* (cap. 3).

2.4 L'ORDINE DEL POLINOMIO

Il modello più semplice di *kernel regression* è chiamato *algoritmo di ordine 0* oppure *Nadaraya-Watson estimator*:

$$\hat{y}_j = \frac{\sum_{i=1}^{nlrn} W(X_{i,d}, X_{j,d}, k, h) * y_i}{\sum_{i=1}^{nlrn} W(X_{i,d}, X_{j,d}, k, h)}, \quad (\text{eq. 2.4.1})$$

\hat{y}_j è il valore di y stimato per il j -esimo punto del *test set*, y_i è il valore osservato di y nell' i -esimo punto di *learning*; \hat{y} potrebbe essere calcolato anche per tutti i valori di LRN e non solo per quelli di *testing*, ma questo richiederebbe più tempo d'analisi.

Questo algoritmo è detto di *ordine 0* perché si serve di una costante (polinomio di ordine 0) per il calcolo di \hat{y} : \hat{y}_j è calcolato, infatti, come una media pesata dei valori di y del *learning set* (y_i).

Dato un *record* j la previsione si ottiene usando i *records* di *learning* che sono più simili a j . La similarità è misurata come la media delle distanze al quadrato definite nell'iperspazio $(X_{i,d}, X_{j,d})$. I regressori *kernel* ottengono la previsione di y_j con la media pesata di y_i dei suoi *nearest neighbors*. I pesi $W(X_{i,d}, X_{j,d}, k, h)$ di ogni *neighbor* vengono determinati con il *kernel* (cap. 2.2 e cap. 2.3).

Un algoritmo più evoluto di *kernel regression* si chiama *algoritmo di ordine 1*, il quale applica, al posto di una semplice media ponderata, un polinomio di primo ordine. Ad una dimensione tale polinomio è una retta:

$$\hat{y}_j = a_1 + a_2 x_i,$$

In p dimensioni si ha invece un iperpiano:

$$\hat{y}_j = a_1 + a_2 x_{i,1} + a_3 x_{i,2} + \dots + a_{p+1} x_{i,p},$$

dall'espressione si deduce che per la previsione nel caso di un *algoritmo di ordine 1* sono necessari $p+1$ coefficienti.

Si possono definire algoritmi di ordine superiore come *l'algoritmo di ordine 2*, basato su un polinomio di secondo ordine, se si considera una sola dimensione si ottiene una parabola:

$$\hat{y}_j = a_1 + a_2 x_{i,1} + a_3 x_{i,2}^2,$$

generalizzando, per p dimensioni si ha invece:

$$\hat{y}_j = a_1 + \sum_{d=1}^p a_{d+1} x_{i,d} + \sum_{d=1}^p \sum_{k=d}^p b_{d,k} x_{i,d} x_{i,k},$$

come si può osservare, più la dimensionalità dello spazio cresce, più il numero di coefficienti necessari per espletare il modello aumenta rapidamente.

In generale il numero di coefficienti richiesti per “fittare” spazi p -dimensionali con algoritmi di *ordine 2* è pari a:

$$1 + p + \frac{p(p+1)}{2}.$$

In applicazioni con una grande componente d'errore nel segnale, come il modellamento dei mercati finanziari, utilizzare algoritmi di ordine superiore a 2 non migliora il risultato finale. In ogni caso, l'aumento dell'ordine accre-

scerebbe la complessità di calcolo e condurrebbe alla perdita di efficienza della metodologia di modellamento, poiché a differenze minime in efficacia corrisponderebbero grandi scostamenti nella complessità computazionale. Si ritiene perciò che sia più che sufficiente considerare solamente gli *algoritmi di ordine 0, 1, 2*.

Una volta scelto l'ordine, i coefficienti vengono determinati per ogni punto del *test set* utilizzando i punti di *learning*. A questo scopo vengono usate tecniche di programmazione lineare, tipicamente il metodo dei minimi quadrati ponderati.

2.5 LA DIMENSIONALITÀ DEL POLINOMIO

Un altro aspetto importante da considerare per applicare la *kernel regression* è la dimensionalità del modello, una caratteristica associata alla selezione dei *candidate predictors*.

La strategia proposta è di prendere ciascun *candidate predictors* singolarmente, sviluppare modelli unidimensionali e abbinare, nelle fasi successive, solo quei *predictors* che riescono a spiegare i dati originali meglio degli altri. Tale procedura viene spesso definita *stepwise*.

Per ridurre la complessità computazionale si deve stabilire il numero massimo di *best candidate predictors* che potranno passare allo stadio successivo. Questo numero è detto “numero di sopravvissuti alla dimensione d ” e lo si indicherà col nome $num_survivors(d)$: solo i $num_survivors(d)$ *candidate predictors* saranno combinati con gli altri per creare gli spazi a $d+1$ dimensioni.

In questo tipo di approccio entra in gioco un altro parametro: la dimensionalità massima da esaminare ($dmax$). Pur essendo questa una valutazione arbitraria la dimensione massima del modello dovrà essere abbastanza elevata da includere il modello buono, sempre se esso esista e sia compatibile con la densità di dati in possesso e le risorse tecnologiche.

In generale, una dimensione massima pari a 10 ($dmax=10$) può essere ritenuta più che sufficiente. In seguito, tuttavia, lo studio della metodologia sarà condotto con $dmax=5$, dimensione ritenuta abbastanza elevata ai fini dimostrativi.

Un altro modo per ridurre la complessità computazionale è quello di vedere il $num_survivors(d)$ come un estremo superiore e stabilire un criterio d’arresto ulteriore che agisca in base a:

- miglioramento (*DELTA*) riscontrato in termini di misura di performance (cap. 2.6) tra un livello (d) e il successivo ($d+1$);
- soglia minima della misura di performance per ritenere un *candidate predictors* un *best predictors*.

Vengono presi in esame successivamente i vari approcci al problema della scelta dei *best predictors* dal punto di vista della complessità computazionale.

Metodo *forward stepwise* semplice

La tecnica *stepwise* semplice ha la particolarità che il valore del parametro $num_survivors$ è pari ad uno in tutti gli stadi:

$$\{num_survivors(d) = 1, \forall d : 1 \leq d \leq d_{max}\}.$$

La tabella qui sotto mostra il numero di spazi totali (S_{tot}) per ogni combinazione di ncp , cioè il numero totale di *candidate predictors* ed il numero totale di dimensioni da analizzare (d_{max}).

ncp	dmax=1	dmax=2	dmax=3	dmax=4	dmax=5
5	5	9	12	14	15
10	10	19	27	34	40
20	20	39	57	74	90
50	50	99	147	194	240
100	100	199	297	394	490
150	150	299	447	594	740
200	200	399	597	794	990

Quando, per esempio, si hanno 5 *candidate predictors* ($n_{cp}=5$) e viene imposto $d_{max}=2$, si deve prima analizzare i *candidate predictors* uno ad uno ($d_{max}=1$), poi si devono considerare e valutare le combinazioni tra il *predictor* sopravvissuto nel primo stadio, poniamo sia x_1 , e i rimanenti, ottenendo i seguenti spazi: (x_1, x_2) , (x_1, x_3) , (x_1, x_4) , (x_1, x_5) . Perciò con $d_{max}=1$ si devono analizzare 5 spazi, con $d_{max}=2$ si devono esaminare $5+4=9$ spazi totali; procedendo in questo modo è stata completata la tabella.

Metodo *forward stepwise* con $num_survivors(d)=10$

Si osservi ora cosa succede se $num_survivors(d)$ è uguale a 10 per tutte le fasi, senza considerare altri criteri d'arresto:

$$\{num_survivors(d) = 10, \forall d / 1 \leq d \leq d_{max}\}.$$

Per calcolare il numero di spazi da analizzare si ricorre ad un'approssimazione data dalla seguente espressione:

$$S(d + 1) \leq num_survivors(d) * (n_{cp} - d),$$

per poi calcolare per ogni valore di d_{max} gli spazi totali da esaminare:

$$S_{tot} = \sum_{d=0}^{d_{max}-1} S(d + 1).$$

Tale espressione non dà il numero esatto di combinazioni da esaminare, ma indica semplicemente il numero massimo di associazioni che si possono avere. Il tutto è illustrato nella tabella seguente:

ncp	dmax-1	dmax-2	dmax-3	dmax-4	dmax-5
5	5	45	75	95	105
10	10	100	180	250	310
20	20	210	390	560	720
50	50	540	1120	1690	2150
100	100	1099	2070	3040	4000
150	150	1640	3120	4590	6050
200	200	2190	4170	6140	8100

Metodo “all combinations”

Se si considerano tutte le possibili combinazioni, senza alcun ricorso ad approcci di tipo *forward stepwise*, il numero di spazi da analizzare esploderebbero secondo la legge del calcolo combinatorio C_d^{ncp} che indica il numero di combinazioni di tutti gli *ncp candidate predictors* presi a gruppi di *d* dimensioni.

Dal calcolo combinatorio:

$$C_d^{ncp} = \frac{ncp!}{d!(ncp - d)!},$$

da cui

$$C_{tot} = \sum_{d=1}^{d \max} C_d^{ncp}.$$

Ricalcolando nuovamente la tabella si ottiene:

nep	dmax=1	dmax=2	dmax=3	dmax=4	dmax=5
5	5	15	25	30	31
10	10	55	175	385	637
20	20	210	1350	6195	21 699
50	50	1275	20 875	251175	2 369 935
100	100	5050	166 750	4 087 975	79 375 495
150	150	11.325	562.625	20.822.900	612.422.930
200	200	20.100	1.333.500	66.018.450	2.601.668.490

Osservando questa ultima tavola si può dedurre che l'esame di tutte le combinazioni dei *candidate predictors* è enormemente costoso ed è proporzionale a *nep* e *dmax*.

È necessaria dunque una strategia di ricerca dei *best predictors* che riesca a limitare le combinazioni da esaminare ad un numero ragionevole. Lo svolgimento di tale ricerca dipende dalla complessità del problema e dal computer utilizzato.

Si è visto che un approccio *forward stepwise* semplice riduce drasticamente le combinazioni possibili, forse troppo. Si ritiene perciò che una strategia di compromesso come l'approccio *stepwise* con l'utilizzo del parametro *num_survivors*, sia da preferirsi nella maggior parte delle situazioni.

2.6 MISURE DI VALUTAZIONE DEL MODELLO

Quando si utilizzano tecniche non parametriche per modellare i dati vengono formulati molti modelli. È importante, quindi, prendere in considerazione alcune misure per poter ordinare i modelli in base alla loro bontà nella fase di ricerca e per poter scegliere il miglior modello nella fase finale.

Un criterio molto utilizzato è la *Variance Reduction (VR)*:

$$VR = 100 * \left(1 - \frac{\sum_{j=nlrn+1}^{ntst} (y_j - \hat{y}_j)^2}{\sum_{j=nlrn+1}^{ntst} (y_j - \bar{y})^2} \right), [-\infty, +100].$$

VR è una buona misura di performance per valutare differenti modelli e per distinguere i buoni *predictors* dai cattivi.

Se $VR = 0$ significa che il modello condurrà a previsioni pari alla media dei valori osservati, se $VR < 0$ l'informazione previsionale è inferiore alla semplice media, mentre, se $VR > 0$ significa che il modello conduce a previsioni migliori che non la semplice media e dunque è un modello da considerare.

VR è una quantità priva di dimensione che non dipende dalla dimensionalità della variabile y , pertanto può essere ritenuto un buon indice di confronto anche tra modelli di natura diversa.

Il problema principale nell'uso di VR è che soffre la presenza degli *outliers*: se esistono anche solo pochi punti di y , che sono molto lontani dalla superficie del modello \hat{y} , questi tenderanno ad avere un effetto sproporzionato sulla valutazione della bontà del modello.

Per ridurre l'effetto degli *outliers*, nella valutazione del modello è bene usare altri indicatori congiuntamente a *VR*, per esempio:

- La *Median Variance Reduction (MVR)*, definita come segue:

$$MVR = 100 * \left(1 - \frac{\text{med}((y_j - \hat{y}_j)^2)}{\text{med}((y_j - \bar{y})^2)} \right), [-\infty, +100].$$

dove *med* è l'operatore mediana.

Questo indicatore offre una valida risposta al problema degli *outliers*, ma aggiunge complessità computazionale al processo, data dalla presenza dell'operatore mediana.

- Il *Fraction Same Sign (FSS)* misura la frazione di punti predetti (\hat{y}_j) il cui segno (+,-) coincide con i valori osservati (y_j) sotto l'ipotesi che \hat{y} e y abbiano media pari a zero, in formula si ha:

$$FSS = 100 * \frac{\sum_{j=nlrn+1}^{ntst} (\hat{y}_j * y_j) > 0}{ntst}, [0, +100].$$

FSS può essere un buon indicatore poiché, se il modello riesce a predire il segno di y_j , si possono sviluppare efficaci strategie di *trading*, in ogni caso non è una misura da sopravvalutare e va usata congiuntamente ad altri indicatori. Il problema nell'uso di *FSS* sta nel fatto che i valori di \hat{y}_j e y_j vicini allo zero vengono trattati allo stesso modo dei valori molto diversi da zero; ciò nonostante, data la facilità di calcolo, è bene usare *FSS* come parametro di *output* in tutti i *report* finali.

- Il *coefficiente di determinazione* al quadrato ($R_{\%}^2$) indica la capacità esplicativa del modello:

$$R_{\%}^2 = 100 * \left(\frac{\sum_{j=nlrn+1}^{ntst} (y_j - \bar{y}) * (\hat{y}_j - \bar{\hat{y}})}{\sqrt{\sum_{j=nlrn+1}^{ntst} (y_j - \bar{y})^2} * \sqrt{\sum_{j=nlrn+1}^{ntst} (\hat{y}_j - \bar{\hat{y}})^2}} \right)^2, \quad [0,+100].$$

Quando si modellano i mercati finanziari per fare *trading* è indispensabile analizzare la *performance* dei *trades* generati dal *trading system* basato sul modello da verificare. Invece di misurare quanto \hat{y}_j si discosta da y_j , con questo criterio si va ad analizzare direttamente i rendimenti. È superfluo considerare questo approccio nella fase di ricerca del modello (selezione dei *best predictors*), ma è d'obbligo nella valutazione dei modelli finali sull'*evaluation set* prima della loro reale applicazione.

Le principali misure che si possono calcolare per esaminare i rendimenti sono:

- la curva dei rendimenti cumulati: il grafico, la media, la varianza, ...;
- il grafico a barre di tutti i *trades* effettuati;
- il massimo profitto;
- la massima perdita;
- la media dei *trades* profittevoli;
- la media dei *trades* negativi;
- l'incidenza di ciascun *trades*, in particolare quello che conduce al massimo profitto, sulla curva dei rendimenti;
- il numero di *trades* vincenti consecutivi;
- il numero di *trades* perdenti consecutivi;
- la frazione di *trades* profittevoli su quelli in perdita;

Queste misure, rispetto alle prime, toccano maggiormente la psicologia dell'*analista-trader*, è bene perciò usarle con cautela ed eventualmente prendere alcuni accorgimenti per sottostimarle, in modo tale da frenare falsi entusiasmi. In alternativa, l'analista dovrebbe tener presente maggiormente le misure delle perdite, che non quelle dei profitti, più la varianza e la media della curva dei rendimenti, che non il profitto finale.

3. KR AD ALTA PERFORMANCE

3.1 DALLE *BANDWIDTH* AL *P-TREE*

Il modellamento di serie storiche finanziarie richiede lo sviluppo di modelli che riescano a trattare migliaia di *records* e centinaia di *candidate predictors* in tempi quanto più brevi tanto più è elevata la frequenza di campionamento dei dati con la quale si opera. È importante, perciò, tener sotto controllo la complessità computazionale del metodo di modellamento adottato.

Con un computer che ha un singolo processore, il tempo totale richiesto per il modellamento è stimato come segue:

$$T_{tot} = S_{tot} * T_{avg},$$

S_{tot} è il numero totale di spazi esaminati e T_{avg} è il tempo medio necessario per studiare ciascun spazio.

Mentre precedentemente sono stati trattati alcuni aspetti relativi al controllo degli spazi totali da analizzare (cap. 2.5), ora l'enfasi cade anche sul tempo medio necessario per analizzare ogni spazio: questi sono entrambi aspetti fondamentali per l'efficienza di un metodo di modellamento.

Applicando un approccio semplice di *kernel regression*, i calcoli richiesti per valutare un singolo spazio sono nell'ordine di $O(nl_{rn} * ntst)$ e il tempo medio necessario tende ad essere proporzionale a tale cifra (cap. 2.3), così che, quando si lavora con decine di migliaia di *records*, il valore di T_{avg} diventa intollerabilmente grande.

Il concetto delle bande descritto nel cap. 2.3 dovrebbe offrire un sensibile miglioramento nella velocità dell'applicazione: la riduzione dei punti di *learning* usati per stimare ogni *test point*, infatti, comporta un numero minore di calcoli da effettuare. Tuttavia anche con l'utilizzo della *bandwidth* vengono computate tutte le distanze tra i punti di *learning* e ciascun punto di

test. Solo successivamente vengono rigettati tutti i punti di *learning* troppo distanti e considerati solo quelli con distanza minore di h (l'ampiezza della *bandwidth*), che verranno pesati con il *kernel* per stimare i valori di y in ogni *test point*.

Un altro inconveniente nell'utilizzo della *bandwidth* è che non sempre la densità dei dati è costante in tutti gli spazi: in uno spazio alcune regioni possono avere una grande concentrazione di *learning points*, mentre altre possono essere scarsamente popolate. Se si considera un'unica banda di ampiezza h per tutti gli spazi, può succedere che alcuni punti di *test* vengono stimati attraverso molti *learning points*, mentre altri punti di *test* da pochi se non da pochissimi, o addirittura non possono essere stimati per mancanza di punti di *learning*, con la conseguenza di avere stime puntuali con varianze diverse.

Si devono quindi sviluppare metodi che riescano rapidamente a localizzare i punti più vicini, senza incorrere nei problemi elencati; questo è un tipico problema di geometria computazionale.

3.2 IL *P-TREE*

Il termine *p-Tree* è usato da *Wolberg* (vedi Appendice e-mail e Bibliografia) per definire una speciale struttura di dati particolarmente utile per modellare efficientemente spazi p -dimensionali con la *kernel regression*.

La struttura è basata su un albero completamente binario con un'altezza prestabilita dall'analista col parametro H (*TREEHEIGHT*), in altre parole è un albero con esattamente 2^H celle "foglie". Ciascun punto di *learning* cade in una di queste *leaf cells* e lo spazio p -dimensionale è partizionato in maniera tale che il numero di punti per cella sia all'incirca costante: se n_{lrn} è un multiplo di 2^H il numero di punti di *learning* per cella è esattamente costante. In definitiva il *p-Tree* serve per raggruppare rapidamente i punti di *learning* più simili in celle per stimare ciascun punto di *testing*.

Se si specifica il numero di *nearest neighbors* da considerare (parametro $numnn$), allora ogni punto di y del *test set* sarà stimato in base a $numnn$ *learning points*; ma per la stima di y non è necessario identificare esattamente i $numnn$ punti più vicini: la scelta dei *learning points* ha una certa influenza sull'accuratezza della predizione, ma se la selezione dei punti più vicini è ragionevolmente approssimata, allora, nella maggior parte dei problemi reali, il calo di bontà delle previsioni è contenuta.

Si può scegliere come *nearest neighbors* solamente i punti contenuti in una *leaf cell*, oppure creare una matrice di adiacenza per identificare tutte le celle adiacenti ad ogni cella, in modo che l'algoritmo possa individuare non solo i punti della cella in cui risiede il punto di test da stimare, ma anche quelli di tutte le celle adiacenti. Così facendo aumenterà la probabilità di trovare tutti i punti di *learning* più vicini a quel punto di test, soprattutto quando si trattano serie con una grande componente di "rumore".

Purtroppo, però, più la dimensionalità del problema cresce, più le celle adiacenti aumentano esponenzialmente e cercare i punti vicini diventa abbastanza costoso. D'altro canto, trascurando alcuni *nearest neighbors*, si ha una minima perdita in termini di accuratezza predittiva: si può ottenere, infatti, una eccellente capacità previsionale anche usando solo i punti nella cella di un dato *test point* (*Fast Kernel Regression*, vedi cap. 4.3).

Tuttavia, tra la ricerca dei *nearest neighbors* effettuata in una sola cella e la ricerca in una cella ed in tutte quelle adiacenti alla stessa, esiste una via intermedia di procedere. Viene introdotto un nuovo parametro chiamato *numcells*, che stabilisce il numero massimo di celle adiacenti da considerare nella ricerca dei *nearest neighbors*. Chiaramente il tempo computazionale cresce al crescere di *numcells*, ma solo fino al valore limite pari al numero massimo di celle adiacenti esistenti, infatti, se fosse specificato un valore di *numcells* superiore al numero totale di celle, l'algoritmo esaminerebbe solo la cella che ospita il *test point* e tutte le celle adiacenti esistenti.

L'introduzione del parametro *numcells* generalizza il concetto di *p-Tree* in quanto consente all'operatore di stabilire quale metodo di ricerca dei *nearest neighbors* scegliere. Se *numcells=1* si avrà il caso più semplice in cui l'algoritmo di ricerca dei *nearest neighbors* opererà solo in una cella, se *numcells* è pari al numero totale di celle adiacenti, l'algoritmo ricerca nella cella del punto di test e in tutte le altre adiacenti, infine, se:

$$1 < \textit{numcells} < \textit{"numero totale di celle adiacenti"}$$

si ha un algoritmo di ricerca intermedio tra i due citati.

Ricapitolando, il *p-Tree* è un semplice modo per creare l'equivalente della *bandwidth* in *p dimensioni*. Una volta che l'albero viene costruito, la cella in cui risiede il punto di test da stimare viene localizzata in un tempo linearmente proporzionale con l'altezza *H* dell'albero ($O(H)$) e la struttura dei dati dovrebbe consentire un immediato accesso a tutti i *learning points*

della cella. Se $numcells > 1$ le celle adiacenti verranno velocemente identificate tramite la matrice di adiacenza e successivamente saranno accessibili i punti di *learning*.

L'uso del *p-Tree* è basato, dunque, sulle seguenti procedure:

- costruzione del *p-Tree* di altezza H utilizzando i nln punti di *learning* che verranno partizionati in 2^H celle;
- se $numcells > 1$, creazione della matrice di adiacenza;
- per ogni punto del test set:
 - localizzazione della cella del *p-Tree* che contiene il *test point*;
 - se $numcells > 1$, ricerca delle $numcells-1$ celle adiacenti;
 - utilizzazione delle $numcells$ celle per cercare i $numnn$ punti di *learning* più vicini;
 - impiego dei $numnn$ punti di *learning* trovati per predire y in ogni punto del *test set*;
- ricorso ai valori stimati di y per determinare un criterio di modellamento appropriato.

Alcune osservazioni sulla complessità del *p-Tree* di altezza H in uno spazio p -dimensionale usando nln *learning points* possono essere le seguenti:

- il massimo numero di classificazioni richieste per procedere dal livello $i-1$ al livello i sono $2^{(i-1)}$;
- per ogni livello $i \leq p$ dovrebbe essere scelta una differente dimensione per la nuova classificazione, in particolare, si dovrebbe scegliere la dimensione con *range* maggiore;
- se $H > p$, per ogni classificazione ai livelli $i > p$, si dovrebbe ordinare in base alla dimensione più lunga;

- gli ordinamenti da fare fino al livello H sono approssimativamente

$$\frac{nlrn}{2^{(H-1)}};$$

Il tempo massimo necessario per tutte le classificazioni può essere stimato considerando il tempo per ordinare n elementi in una lista: $O(n \cdot \log_2(n))$. Si può così sviluppare un'espressione generale che indichi il tempo massimo per tutti gli ordinamenti fino al livello H :

$$T_{sort} \leq C * \sum_{i=1}^H nlrn * (\log_2(nlrn) - i + 1),$$

dove C è una costante che dipende dalla potenza del computer.

L'espressione è una disuguaglianza perché non tutti gli ordinamenti devono necessariamente essere attuati.

La sommatoria si può sviluppare e la disequazione precedente si trasforma nella seguente:

$$T_{sort} \leq C * nlrn * (H * \log_2(nlrn) - (H - 1) * H / 2).$$

Questa espressione evidenzia che il limite massimo di tempo richiesto per effettuare tutte le classificazione di tutti i livelli è inferiore ad H volte il tempo necessario ad un unico ordinamento di tutti gli $nlrn$ elementi. Inoltre, il tempo reale potrebbe essere significativamente inferiore a questo limite massimo nel caso certi ordinamenti in alcuni livelli risultassero non essere necessari: un nuova classificazione è necessaria, infatti, solo se l'ordinamento è basato su una dimensione differente rispetto alla dimensione scelta nel livello precedente.

3.3 L'UTILIZZO DEL P-TREE

Dopo aver partizionato i punti di *learning*, lo scopo del *p-Tree* è quello di identificare rapidamente i *nearest neighbors* di ciascun *testing points* al fine di stimare i valori di y del *test set* (\hat{y}_j). L'albero è usato per determinare le “celle foglie” di ogni punto di test, per alberi di altezza H sono indispensabili una sequenza di H istruzioni condizionali per localizzare la cella appropriata.

È buona abitudine non tentare di stimare valori di y del *test set* che sono *out-of-range*, cioè i punti che cadono fuori dall'intervallo che ha per estremi il valore minimo ed il valore massimo del *learning set*. Dal momento che la superficie che deve essere modellata può risultare altamente non lineare, considerando i punti *out-of-range* l'estrapolazione potrebbe condurre ad errori elevati.

Concettualmente lo scopo è di utilizzare serie storiche per tentare di costruire dei modelli previsionali, d'altro canto però, i punti del *database* storico che sono *out-of-range* possono comportarsi in maniera molto diversa dai punti storici. Nel valutare gli spazi è ragionevole allora scartare tutti i punti del *test set* che si rivelano *out-of-range*.

Informazioni aggiuntive per la localizzazione delle “celle foglie” possono essere ricavate mediante la matrice di adiacenza. Se, per esempio, si deve trovare la cella adiacente più vicina alla cella contenente il *test point*, prima di tutto si dovrà ricorrere alla matrice di adiacenza per determinare quali siano le celle adiacenti e poi individuare quali di queste sia la più vicina, dove col termine “più vicina” si può intendere:

- la cella che contiene i punti più vicini al *test point* sulla superficie (o lato se il *p-Tree* tratta solo due dimensioni: $p=2$);
- la cella con il baricentro più vicino al *test point*.

Le due definizioni possono condurre a diverse conclusioni, ma tali differenze non sono rilevanti, in quanto, come si è detto, ciò che interessa è una soluzione approssimata. Tuttavia, più la dimensionalità del modello cresce, più il compito di trovare i punti più vicini diventa difficile e per ridurre la complessità computazionale è preferibile usare la tecnica del baricentro.

Il *p-Tree* viene adoperato per fare previsioni sulla variabile y nel *test set*, a questo punto si può scegliere tra un vasto assortimento di algoritmi come l'*ordine 0*, l'*ordine 1*, l'*ordine 2* introdotti nel cap. 2.4, e considerare differenti opzioni di apprendimento: *learning set* statico o dinamico (cap. 2.1).

Dopo aver calcolato \hat{y} nel *test set*, si calcolano alcune misure di performance, prima fra tutte *VR* (cap. 2.6), per valutare il modello. Si procede così, sia nelle fasi di ricerca dei *best predictors* con la tecnica *stepwise* (cap. 2.5), sia nella fase di validazione del modello.

Il *p-Tree* viene allo stesso modo utilizzato nello stadio in cui si effettuano le previsioni vere e proprie, come scritto da *Wolberg* in risposta ad una mia *e-mail* (vedi Appendice e-mail).

3.4 COMPLESSITÀ COMPUTAZIONALE

Sul tempo associato al processo di modellamento si è già parlato nella sezione 3.1 in cui si affermava che il tempo necessario per sviluppare un set di modelli è proporzionale al prodotto tra il numero di spazi esaminati e il tempo medio richiesto ad ogni spazio:

$$T_{tot} = S_{tot} * T_{avg}.$$

Questa formula è basata sull'ipotesi che il computer usato abbia un singolo processore. In ogni caso, si possono estendere le nozioni già enunciate e quelle che verranno presentate successivamente all'eventualità che il computer utilizzato possieda più processori. Oggi infatti, si possono trovare in commercio calcolatori con 2, 4 e perfino 8 processori in parallelo. In tal caso, teoricamente, il tempo totale necessario all'elaborazione (T_{tot}) si ridurrebbe di un fattore pari a N , dove N è il numero di processori disponibili.

Il tempo computazionale, inoltre, può essere condizionato da molteplici varianti come per esempio la quantità ed il tipo di memoria RAM, la presenza o meno di coprocessori matematici, il tipo di sistema operativo e l'integrità del sistema.

La trattazione di tutti questi fattori esula, però, lo scopo generale del paragrafo, che è semplicemente quello di stimare approssimativamente il tempo di calcolo della *kernel regression*, senza aver la pretesa di basare le strategie di modellamento su equazioni teoriche inerenti la complessità computazionale: gran parte del lavoro è empirico e la teoria funge solo da supporto.

Nell'applicare modelli di *kernel regression* ad alta performance con l'utilizzo del *p-Tree*, il tempo medio per spazio (T_{avg}) può essere considerato

come la somma di due componenti base: il tempo di preparazione (*preparation time*) ed il tempo di elaborazione (*run time*):

$$T_{avg} = T_{prep} + T_{run}.$$

Il tempo di preparazione: T_{prep}

Il tempo di preparazione è formato da tre componenti: il tempo indispensabile per gli ordinamenti o *sort time* (T_{sort}), il tempo necessario per sviluppare la matrice di adiacenza (T_{adj}) se vengono usate le celle adiacenti per trovare i *nearest neighbors*, ed infine il tempo richiesto per salvare il registro (*bookkeeping*) di tutti i nodi del *p-Tree* (T_{book}):

$$T_{prep} = T_{sort} + T_{adj} + T_{book}.$$

- Di T_{sort} si è già parlato nella sezione 3.2 ed è proporzionale ad H ed a $n \ln n$:

$$T_{sort} \leq C * n \ln n * (H * \log_2(n \ln n) - (H - 1) * H / 2),$$

- T_{adj} è il tempo dovuto per la costruzione della matrice di adiacenza ed è pari a zero nell'unico caso in cui viene usata solo la cella dove risiede il punto di test per trovare i punti più vicini. Tuttavia, se la matrice di adiacenza è richiesta ($numcells > 1$), T_{adj} è un termine importante e per elevati valori di H può diventare la quantità dominante di T_{prep} . T_{adj} infatti, è proporzionale al quadrato del numero delle “celle foglie” (2^{2H}), in altre parole, al crescere del valore H di un'unità, T_{adj} aumenta di un fattore pari a quattro:

$$T_{adj} = C_{adj} * 2^{2H},$$

Se il numero di *learning points* per ogni cella (*ppc*) è tenuto costante, allora si può relazionare H con nln :

$$ppc = \frac{nln}{2^H},$$

moltiplicando entrambe le proposizioni per 2^H e poi facendo il logaritmo in base due, si scopre che H cresce proporzionalmente a $\log_2(nln)$, si otterrà:

$$H = \log_2(nln) + ppc, \text{ con } ppc \text{ costante}$$

e dunque

$$T_{adj} = C_{adj} * 2^{2(\log_2(nln) + ppc)},$$

confrontando questa espressione con quella del *Sort Time*:

$$T_{sort} \leq C * nln * (H * \log_2(nln) - (H - 1) * H / 2),$$

se ne deduce che per grandi valori di nln , T_{sort} diventa piccolo rispetto a T_{adj} , sempre che venga calcolata la matrice di adiacenza ($T_{adj} > 0$).

– T_{book} è, invece, proporzionale al numero di nodi presenti nel *p-Tree*:

$$T_{book} = C_{book} * (2 * 2^H - 1),$$

ma se i punti per cella sono costanti si ha:

$$T_{book} = C_{book} * (2 * 2^{(\log_2(nln) + ppc)} - 1),$$

dunque T_{book} aumenta meno velocemente rispetto a T_{sort} e T_{adj} .

Sintetizzando, raddoppiando $nlnr$, per mantenere costante ppc si dovrà aumentare H di un'unità, così facendo T_{book} crescerà di un fattore circa uguale a 2, T_{sort} aumenterà di un fattore leggermente più grande di 2 e T_{adj} si prolungherà di un fattore pari a 4. Allora per elevati valori di $nlnr$ il termine dominante nella complessità computazionale di pre-analisi (T_{prep}) è T_{adj} se $T_{adj} \neq 0$.

Il tempo di elaborazione: T_{run}

Il tempo di elaborazione T_{run} è proporzionale al numero di *test points*:

$$T_{run} = C_{run} * ntst,$$

dove C_{run} è una costante che indica il tempo di elaborazione per un singolo punto di test e dipende dai parametri che si utilizzano nel modellamento.

I principali elementi che concorrono a determinare C_{run} sono quattro:

$$C_{run} = C_{cell} + C_{search} + C_{weight} + C_{solve},$$

- C_{cell} è il tempo impiegato per localizzare la cella in cui risiede il *test point*, è proporzionale ad H ed ha una durata molto breve.
- C_{search} rappresenta il tempo necessario per cercare i *numnn nearest neighbors*, può essere pari a zero oppure può rappresentare la componente dominante di C_{run} . È uguale a zero se vengono usati solo i *learning points* della cella in cui risiede il punto di test per stimare \hat{y}_j . Se, invece, viene eseguita una ricerca, C_{search} è proporzionale al prodotto tra il numero di celle incluse nella ricerca (*numcells*) e i punti di ogni cella (ppc), inoltre dipende dal numero di *nearest neighbors* da trovare (*numnn*).

- C_{weight} è legato al tempo indispensabile per determinare i pesi da assegnare ad ogni *nearest neighbors*, è proporzionale a $numnn$ a meno che tutti i punti non siano equamente pesati, in tal caso C_{weight} risulta pari a zero.
- C_{solve} rappresenta il tempo richiesto per calcolare i valori di \hat{y}_j con l'algoritmo della *kernel regression* e dipende dall'ordine e dalla dimensionalità del polinomio utilizzato. Se l'ordine è 0 vi è un'unica equazione ad un'incognita, se l'ordine è 1 le equazioni da trovare sono $p+1$, mentre se l'ordine è 2 il numero di equazioni è $1 + p + (p + 1) * p / 2$. Il tempo necessario per risolvere le equazioni è approssimativamente proporzionale al numero di equazioni elevato alla terza potenza ($num_equations^3$), mentre il tempo necessario all'ultimo stadio (la stima di \hat{y}_j) è minore rispetto a quello richiesto per risolvere le equazioni ed è proporzionale a $num_equations$.

Da quanto detto sulle componenti di C_{run} , le condizioni da rispettare per ridurre al minimo il tempo di elaborazione sono:

- ricorrere solo ai *learning points* localizzati nella cella in cui è caduto il punto di test con la conseguenza che $C_{search}=0$;
- pesare tutti i punti equamente in modo che $C_{weight}=0$;
- quando dei punti di *learning* vengono aggiunti o tolti da una cella (*learning set* dinamico: *growing e moving option*), ricalcolare i coefficienti solo dopo che un nuovo punto di test viene posto nella cella.

Quando si usano queste strategie, il tempo di elaborazione T_{run} si riduce notevolmente e di solito diventa più breve del tempo di preparazione T_{prep} , che a sua volta si abbrevia per il fatto che non è necessario definire la matrice di adiacenza ($T_{adj} = 0$). Questo veloce criterio di modellamento è chiama-

to *Fast Kernel Regression* ed è utilizzato soprattutto quando tra efficacia ed efficienza del modello si predilige l'efficienza, quando cioè, a piccoli scostamenti nel miglioramento della *performance* del modello, si preferiscono più elevati scostamenti in termini di riduzione di tempo nel raggiungere i risultati. La *Fast Kernel Regression* sembra dunque un metodo ideale nel caso di operatività *intraday* nei mercati finanziari: un modello che apprende ed elabora velocemente i dati di input può essere in grado di fare previsioni con un ritardo sostenibile rispetto alla frequenza d'aggiornamento dei dati (15', 10', 5', 1', *Tick*).

3.5 IL TEMPO PESA I DATI

Nelle applicazioni finanziarie, l'ordine dei dati, segnato dalle date, se si stanno trattando dati giornalieri, oppure dalla data e l'orario, se si considerano dati *intraday*, è un fattore rilevante che i modelli predittivi dovrebbero tener presente. La dimensione tempo, dunque, introduce un altro livello di complessità nell'analisi e fa riaffiorare alcune domande: quali punti del *learning set* vanno a stimare ciascun punto del *test set*? Quanta importanza si deve dare ai *records* più recenti e quanta a quelli più datati? E quando un *records* si può dire che è datato?

Quando si trattano serie storiche, la dimensione temporale è importante, in particolar modo, quando si modellano dati finanziari è ragionevole assumere che i *records* più recenti siano maggiormente rilevanti rispetto ai dati di un periodo antecedente. Se bisogna prevedere il valore di y in un dato momento temporale si devono considerare maggiormente i *learning points* più vicini a quel periodo, è assurdo, infatti, ottenere dei buoni modelli predittivi se le stime dei valori vengono basate equamente su punti datati e punti recenti del *learning set*.

Un semplice criterio per dare più importanza ai punti più attuali è quello di pesare i dati in base al tempo, così facendo il *kernel* esponenziale (eq. 2.2.2) deve essere modificato in questo modo:

$$W(X_{i,d}, X_{j,d}, t, k, \alpha) = e^{-(kD_{i,j}^2 + \alpha t)},$$

In questa equazione t è la differenza temporale tra il tempo associato al j -esimo punto di test e l' i -esimo *learning point*, α è una costante temporale definita dall'analista in base alla percezione soggettiva che ha sul peso dato

dalla variabile tempo: se il tempo viene misurato in unità di un giorno e l'analista ritiene che le informazioni apprese da dati vecchi di un anno (365 giorni) debbano ricevere metà del peso delle informazioni correnti, allora α sarà calcolato risolvendo l'equazione:

$$1/2 = e^{-365\alpha}$$

da cui

$$\alpha = \ln 0.5 / -365 = 0.0019 .$$

3.6 DAY TRADING ED INTRADAY TRADING

L'importanza della variabile tempo può variare a seconda del tipo di operatività che si ha intenzione di effettuare sui mercati finanziari. Esistono, infatti, due principali tipi di operatività: il *day trading* e l'*intraday trading*.

La differenza sostanziale tra i due tipi di *trading* sta nel fatto che, mentre nel *day trading* ad ogni intervallo (un intero giorno borsistico) vengono evidenziati i quattro valori fondamentali (apertura, chiusura, massimo, minimo) e si ha che l'estremità dell'intervallo corrisponde sempre alla fine delle contrattazioni; l'*intraday trading* è, invece, caratterizzato dal fatto che alla fine di ogni intervallo (60', 15', 10', 5', 1', tick) se ne sussegue un altro di contrattazioni, eccetto che dopo la chiusura della giornata borsistica e prima dell'apertura della successiva. Questo fa sì che si creino dei *gap* tali da non poter fare a meno di spiegare la variabile tempo con due variabili: i giorni borsistici (data: giorno, mese, anno) e gli intervalli minori (orario: ora e minuti).

La dimensione tempo, dunque, introduce un altro livello di complessità nell'analisi quando si trattano serie storiche *intraday*, mentre potrebbe essere addirittura sostituita con un contatore nel caso del *day trading*.

4. STUDIO DI FATTIBILITÀ

4.1 PRESENTAZIONE DELLA MATRICE DATI

Lo studio di fattibilità è stato condotto inizialmente con “dati artificiali” creati mediante il seguente *script* in linguaggio *Matlab*:

```

%% M-file used to create the FKR data file.
%% Columns of DATA: [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10;...
%% y,y50,y25,y10,y05; count; date; times].

% Number of Records
nrec=15000;

% Number of Candidate Predictors
ncp=10;

% Creates an n*(ncp+8) zero filled matrix
DATA=zeros(nrec,ncp+8);

% Fills columns 1 thru ncp with Gaussian(0,1) random noise
DATA(:,1:ncp)=random('Normal',0,1,nrec,ncp);

% The pure signal y is a 3D non-linear function
y1=exp(-((DATA(:,2)+0.5).^2))-exp(-((DATA(:,2)-0.5).^2));
y2=exp(-((DATA(:,5)+0.5).^2))-exp(-((DATA(:,5)-0.5).^2));
y3=exp(-((DATA(:,9)+0.5).^2))-exp(-((DATA(:,9)-0.5).^2));
y=y1+y2+y3;
DATA(:,ncp+1)=y;

% Increase all x2's values by 10
DATA(:,2)=DATA(:,2)+10;

% Scale values of x5 up by a factor of 1000
DATA(:,5)=1000*DATA(:,5);

% Scale values of x9 down by a factor of 1000
DATA(:,9)=DATA(:,9)/1000;

% Generates a vector with nrec random numbers from the...
% continuous uniform distribution [-1,1]
noise=random('Uniform',-1,1,nrec,1);

% Noise Standard Deviation
sdnoise=std(noise);

% Standard Deviation of the pure signal y
sdy=std(y);

% MAX_VR=50% (Signal=50%,Noise=50%)

```

```
y50=y+(sdy/sdnoise)*sqrt(0.50/0.50)*noise;
DATA(:,ncp+2)=y50;

% MAX_VR=25% (Signal=25%,Noise=75%)
y25=y+(sdy/sdnoise)*sqrt(0.75/0.25)*noise;
DATA(:,ncp+3)=y25;

% MAX_VR=10% (Signal=10%,Noise=90%)
y10=y+(sdy/sdnoise)*sqrt(0.90/0.10)*noise;
DATA(:,ncp+4)=y10;

% MAX_VR= 5% (Signal=5%,Noise=95%)
y05=y+(sdy/sdnoise)*sqrt(0.95/0.05)*noise;
DATA(:,ncp+5)=y05;

% Counter
count=(1:nrec)';
DATA(:,ncp+6)=count;

% Dates: DATA(:,ncp+7)=date;
% Times: DATA(:,ncp+8)=times;
```

Lo *script* esposto genera una matrice (15.000 x 17): le prime 10 colonne contengono i candidate predictors (\underline{x}), costruiti con un generatore di numeri casuali di distribuzione gaussiana, in particolare x_2 , x_5 , x_9 , sono il risultato anche di ulteriori trasformazioni eseguite per dimostrare la capacità della *kernel regression* di individuare i *best predictors* indipendentemente dalla loro grandezza e quindi senza il bisogno di alcuna standardizzazione dei dati.

Le colonne dalla 11° alla 15° contengono la variabile dipendente y con differenti frazioni di segnale e rumore:

- la 11° è una serie storica non lineare a tre dimensioni costruita utilizzando i *predictors* x_2 , x_5 , x_9 in modo da contenere il segnale puro (100% segnale, 0% rumore);
- la 12° ha il 50% di segnale e il 50% di rumore;
- la 13° ha il 25% di segnale e il 75% di rumore;
- la 14° ha il 10% di segnale e il 90% di rumore;
- ed infine la 15° colonna contiene la y con il 5% di segnale e il 95% di rumore.

4.2 PRESENTAZIONE DEI PARAMETRI PRINCIPALI

NOME	VALORI DEFAULT	VALORI LIMITE	BREVE DESCRIZIONE
BEST_MODELS	1	[0,5]	Indica quanti modelli verranno generati e salvati nel file di output
DELTA	-100	Dipende dal <i>Survivor concept</i> (cap. 2.5)	Incremento minimo nella performance del modello per passare allo stadio successivo
DMAX	1	[1,10]	Dimensione massima del modello (cap. 2.5)
DMIN	1	[1,5]	Dimensione minima del modello
FAST	0	1: FKR di Ordine 0 2: FKR di Ordine 1 3: FKR di Ordine 2	Se $FAST \neq 0$ si ha la <i>Fast Kernel Regression</i> : $NUMK=1$, $K(1)=1$, $NUMCELL=1$, $FIT_ORDER = \{0, 1, 2\}$. (cap. 3.4)
ORDER	1	{ 0, 1, 2 }	Ordine del polinomio (cap. 2.4)
GROWTH_FACTOR	2	Deve essere > 1 se LTYPE è G, è irrilevante se LTYPE = 'S', 'M'	Stabilisce il numero massimo di <i>learning points</i> che possono essere aggiunti nell'aggiornamento del <i>learning set</i>
HALF_LIFE	0	Vedi "Il tempo pesa i dati" (cap. 3.5)	Peso = $[\ln(HALF_LIFE) / -365]$ (cap.3.5)
K(i)	1	[0,1]	Valori degli <i>Smoothing Parameters</i> (cap. 2.2)
H(TREEHEIGHT)	0	$[0, \log_2(nln / ppc)]$	Altezza del <i>p-Tree</i> (cap. 3.2)
LTYPE	S	('G', 'M', 'S')	Opzioni di aggiornamento del <i>Learning Set</i> : <i>Growing, Moving, Static</i> . (cap. 2.1)
MC	VR	('VR', 'FSS', 'CC')	Misure di performance adottate come criterio di modellamento (cap. 2.6)
nevl	0	$nevl \leq nrec - nln - ntst$	Numero totale di punti nell' <i>Evaluation Set</i> (cap. 2.1)
nln	$nrec - ntst$	$[2^H, nrec]$	Numero totale di punti nel <i>Learning Set</i> (cap. 2.1)
ncol	Da file Input		Numero totale di colonne della matrice di input
nrec	Da file Input		Numero totale di <i>records</i> della matrice di input
ntst	Non specificato	$0 < ntst \leq nrec - nln$	Numero totale di punti nel <i>Test Set</i> (cap. 2.1)
NUMCELLS	1	Si auto incrementa se NUMNN è troppo elevato (cap. 3.2)	Numero di celle in cui vengono cercati i <i>nearest neighbors</i> (se > 1 : costruzione matrice di adiacenza e ricerca delle celle adiacenti)
NUMNN	nln / H^2	[1,nln]	Numero di <i>nearest neighbors</i> da cercare per calcolare la <i>Kernel Regression</i> (cap. 3.2)

Kernel Regression: Applicazione alla Previsione del Fib30

ncp	DMAX	[1, ncol)	È il numero totale di <i>Candidate Predictors</i>
NUMK	1	[1,20]	Numero totale di <i>Smoothing Parameters</i>
SURVIVE_NUM(dim)	0	0 significa che tutti gli spazi alla dimensione dim « sopravvivono »	Indica il numero massimo di spazi alla dimensione dim che vengono salvati e combinati per la dimensione $dim+1$ (cap. 2.5)
POINTLEAF	50	[5,NLRN]	Media di punti ospitati in ogni cella foglia (cap. 3.2)
PRINT	0	[0,5]	Grado di dettaglio nel file di output
STARTEVL	STARTTS T+NTST	Dopo i records di test	Inizializzazione del contatore dei records di EVL
STARTLRN	1	$STARTLRN+nln - 1 \leq nrec$	Inizializzazione del contatore dei records di LRN
STARTTST	$nrec - ntst - nevl + 1$	$STARTTST + ntst - 1 \leq nrec$	Inizializzazione del contatore dei records di TST
STARTCP	1	$X(i) = \text{colonna}(i + STARTCP - 1)$	Inizializzazione del contatore dei Candidate Predictors
TIMECOL	Non specificato	[1,NCOL]	Indice della colonna del tempo
TIMING	0	0: NON inserisce il tempo del processo nel report 1: inserisce il tempo del processo nel report	Rende facoltativo l'inserimento nel report del tempo computazionale, che il processo ha richiesto, espresso in secondi
YCOL	Non specificato	[1,ncol]	Indice della colonna della serie storica y oggetto della previsione

4.3 DESCRIZIONE DEI RISULTATI OTTENUTI

Sono state eseguite numerose prove allo scopo di dimostrare l'abilità della *kernel regression* di trovare un modello per dati non lineari e multidimensionali. In particolare, l'analisi descritta di seguito illustra le *performance* dell'applicazione nel *fittare* una variabile dipendente che ha il 10% di segnale ed il 90% di rumore (colonna 14°).

Si è scelto di mostrare i risultati di tale variabile nell'ipotesi che la serie finanziaria del *Fib30* possa avere circa questa frazione di segnale.

Si utilizzerà come criterio di modellamento (*MC*) la misura *VR* (*Variance Reduction*): la migliore *performance* che si potrà ottenere con qualsiasi tecnica di modellamento di una variabile dipendente con il 10% di segnale è $VR=10\%$.

Non verranno mostrate tutte le analisi effettuate sulle varie serie con differenti frazioni di segnale, basti considerare che più la componente di segnale è forte, più il *p-Tree* riesce a classificare nettamente i vari *pattern* del *learning set* (*nearest neighbors*) nelle celle. In altre parole, più grande è la frazione di segnale più *pattern* si riusciranno ad individuare, dunque più celle si potranno costruire e la regressione potrà stimare al meglio i valori anche solo con pochi *nearest neighbors*, al contrario, se vi è molto rumore, la regressione dovrà considerare abbastanza *nearest neighbors* da riuscire a compensare il rumore di ogni punto di *learning*.

Si è proceduto inizialmente all'analisi della *Fast Kernel Regression* (*FKR*): *kernel regression* che utilizza il *p-Tree* per cercare i *nearest neighbors* (*NUMNN*) senza dover ricorrere al calcolo delle distanze e che successivamente stima y utilizzando solo i punti di *learning* di un'unica cella

($NUMCELL=1$) pesandoli equamente con un unico *kernel* $\{NUMK=1, K(1)=1\}$.

La tabella mostra i risultati ottenuti con tale applicazione.

Fast Kernel Regression: NUMK=1, K(1)=1, NumCell=1, NumNN (irrelevante)											
p-TREE			ORDER 0			ORDER 1			ORDER 2		
TREE HEIGHT	LEAF CELL	POINT LEAF	VR	X(i)	Time (sec.)	VR	X(i)	Time (sec.)	VR	X(i)	Time (sec.)
1	2	5000,0	3,189	5	0	3,186	5	1	3,497	9	2
2	4	2500,0	6,215	5,9	3	6,168	5,9	4	6,678	5,9	5
3	8	1250,0	8,505	2,5,9	6	8,287	2,5,9	8	8,878	2,5,9	10
4	16	625,0	8,613	2,5,9	7	8,341	2,5,9	9	7,904	2,5,9	11
5	32	312,5	8,406	2,5,9	8	8,426	2,5,9	10	6,470	2,5,9	12
6	64	156,3	8,023	2,5,9	9	7,673	2,5,9	11	3,701	2,5	13
7	128	78,1	7,833	2,5,9	10	5,229	2,5,9	12	1,307	9	14
8	256	39,1	6,481	2,5,9	11	0,058	2,5,9	13	-2,921	9	15
9	512	19,5	5,336	2,5,9	12	-5,407	5	14	-11,566	4	16
10	1024	9,8	0,516	2,5,9	13	-16,073	5	15	-	-	-
11	2048	4,9	-6,791	2,5,9	14	-	-	-	-	-	-
12	4096	2,4	-26,811	2,5,9	17	-	-	-	-	-	-
13	8192	1,2	-59,821	2,5,9	19	-	-	-	-	-	-

Osservando la tabella si può vedere che il massimo valore di *VR* si è ottenuto con l'algoritmo di ordine 2 (*ORDER 2*) e altezza del *p-Tree* pari a 3 ($H=3$), ciò nonostante l'algoritmo di ordine 0 (*ORDER 0*) conduce a valori di *VR* abbastanza elevati in minor tempo computazionale e meno dipendenti dall'altezza dell'albero ($H=\{3,4,5\}$), inoltre sembra essere l'algoritmo che fallisce meno volte nell'individuazione dei tre *best predictors* (X_2, X_5, X_9).

Di seguito si mostra, a titolo d'esempio, il report di output della *Fast Kernel Regression* con $H=4$ e $ORDER=0$; i valori asteriscati corrispondono a parametri tenuti costanti in tutte le analisi eseguite.

```

DMIN - min number of dimensions      :      1 (*)
DMAX - max number of dimensions       :      3 (*)
NCP  - number of candidate predictors  :     10 (*)
NCOL - number of columns of data      :     18 (*)
    
```

Dino Monico

NREC - total number of records : 15000 (*)
LTYPE - Learning set type (G, M, S) : S (*)
G: growing, M: moving, S: static
NUMCELL- Number of cells searched for NUMNN : 1 (*)
NUMNN - Irrelevant because FAST mode specified
STARTTCP- starting variable : 1 (*)
DELTA - Min incremental change : -100 (*)
NODATA - For Ycalc : 999 (*)

Kernel Regression parameters:

MC - 1=VR, 2=CC, 3=CCO, : 1 (*)
ORDER - 0 is average, 1 & 2 are surfaces : 0
FAST option used: Ycalc is cell average
NUMK - Number of smoothing parameters : 1 (*)
K(1) - Smoothing parameter 1 : 1.00 (*)
YCOL - Column of dependent variable : 14 (*)

Tree parameters:

BUCKETSIZE - design number per cell (computed) : 625
TREEHEIGHT - tree parm : 4
Computed number of cells : 31
Computed number of leaf cells : 16
Computed avg bucket size : 625.0
Computed 2 Sigma limit for F Stat : 1.73

Data set parameters:

NLRN - number of learning records : 10000 (*)
NTST - number of test records : 5000 (*)
NEVL - number of evaluation records : 0 (*)
GAP - gap records between data sets : 0 (*)
STARTLRN - starting learning record : 1 (*)
STARTTST - starting test record : 10001 (*)

Dim: 1	Var Red:	-0.077	FracSS:	0.495	K:	1.00	X(1) =	1
Dim: 1	Var Red:	2.758	FracSS:	0.547	K:	1.00	X(1) =	2
Dim: 1	Var Red:	-0.177	FracSS:	0.504	K:	1.00	X(1) =	3
Dim: 1	Var Red:	-0.326	FracSS:	0.508	K:	1.00	X(1) =	4
Dim: 1	Var Red:	3.482	FracSS:	0.552	K:	1.00	X(1) =	5
Dim: 1	Var Red:	-0.090	FracSS:	0.501	K:	1.00	X(1) =	6
Dim: 1	Var Red:	-0.439	FracSS:	0.491	K:	1.00	X(1) =	7
Dim: 1	Var Red:	-0.013	FracSS:	0.505	K:	1.00	X(1) =	8
Dim: 1	Var Red:	3.579	FracSS:	0.549	K:	1.00	X(1) =	9
Dim: 1	Var Red:	-0.009	FracSS:	0.509	K:	1.00	X(1) =	10

Ordered results for 1 dimensional models:

Total number of combinations:	10 (*)
Number tested	: 10 (*)
survivenum(1)	: 5 (*)
Number of survivors	: 5

Var Red:	3.579	FracSS:	0.549	X:	9
Var Red:	3.482	FracSS:	0.552	X:	5
Var Red:	2.758	FracSS:	0.547	X:	2

Kernel Regression: Applicazione alla Previsione del Fib30

Var Red: -0.009 FracSS: 0.509 X: 10
 Var Red: -0.013 FracSS: 0.505 X: 8

Dim: 2	Var Red:	2.971	FracSS:	0.549	K:	1.00	X(1) = 1	X(2) = 9
Dim: 2	Var Red:	5.267	FracSS:	0.543	K:	1.00	X(1) = 2	X(2) = 9
Dim: 2	Var Red:	3.064	FracSS:	0.550	K:	1.00	X(1) = 3	X(2) = 9
Dim: 2	Var Red:	3.184	FracSS:	0.550	K:	1.00	X(1) = 4	X(2) = 9
Dim: 2	Var Red:	6.172	FracSS:	0.553	K:	1.00	X(1) = 5	X(2) = 9
Dim: 2	Var Red:	3.013	FracSS:	0.549	K:	1.00	X(1) = 6	X(2) = 9
Dim: 2	Var Red:	2.672	FracSS:	0.549	K:	1.00	X(1) = 7	X(2) = 9
Dim: 2	Var Red:	3.004	FracSS:	0.549	K:	1.00	X(1) = 8	X(2) = 9
Dim: 2	Var Red:	3.046	FracSS:	0.549	K:	1.00	X(1) = 9	X(2) = 10
Dim: 2	Var Red:	3.073	FracSS:	0.552	K:	1.00	X(1) = 1	X(2) = 5
Dim: 2	Var Red:	5.644	FracSS:	0.552	K:	1.00	X(1) = 2	X(2) = 5
Dim: 2	Var Red:	3.270	FracSS:	0.552	K:	1.00	X(1) = 3	X(2) = 5
Dim: 2	Var Red:	3.106	FracSS:	0.552	K:	1.00	X(1) = 4	X(2) = 5
Dim: 2	Var Red:	3.072	FracSS:	0.552	K:	1.00	X(1) = 5	X(2) = 6
Dim: 2	Var Red:	2.984	FracSS:	0.552	K:	1.00	X(1) = 5	X(2) = 7
Dim: 2	Var Red:	3.117	FracSS:	0.552	K:	1.00	X(1) = 5	X(2) = 8
Dim: 2	Var Red:	3.069	FracSS:	0.552	K:	1.00	X(1) = 5	X(2) = 10
Dim: 2	Var Red:	2.258	FracSS:	0.547	K:	1.00	X(1) = 1	X(2) = 2
Dim: 2	Var Red:	2.291	FracSS:	0.547	K:	1.00	X(1) = 2	X(2) = 3
Dim: 2	Var Red:	2.228	FracSS:	0.546	K:	1.00	X(1) = 2	X(2) = 4
Dim: 2	Var Red:	2.331	FracSS:	0.547	K:	1.00	X(1) = 2	X(2) = 6
Dim: 2	Var Red:	2.039	FracSS:	0.548	K:	1.00	X(1) = 2	X(2) = 7
Dim: 2	Var Red:	2.176	FracSS:	0.547	K:	1.00	X(1) = 2	X(2) = 8
Dim: 2	Var Red:	2.304	FracSS:	0.547	K:	1.00	X(1) = 2	X(2) = 10
Dim: 2	Var Red:	-0.006	FracSS:	0.506	K:	1.00	X(1) = 1	X(2) = 10
Dim: 2	Var Red:	0.025	FracSS:	0.511	K:	1.00	X(1) = 3	X(2) = 10
Dim: 2	Var Red:	-0.180	FracSS:	0.506	K:	1.00	X(1) = 4	X(2) = 10
Dim: 2	Var Red:	0.054	FracSS:	0.507	K:	1.00	X(1) = 6	X(2) = 10
Dim: 2	Var Red:	-0.372	FracSS:	0.486	K:	1.00	X(1) = 7	X(2) = 10
Dim: 2	Var Red:	-0.057	FracSS:	0.504	K:	1.00	X(1) = 8	X(2) = 10
Dim: 2	Var Red:	-0.162	FracSS:	0.503	K:	1.00	X(1) = 1	X(2) = 8
Dim: 2	Var Red:	-0.121	FracSS:	0.506	K:	1.00	X(1) = 3	X(2) = 8
Dim: 2	Var Red:	-0.031	FracSS:	0.515	K:	1.00	X(1) = 4	X(2) = 8
Dim: 2	Var Red:	-0.208	FracSS:	0.496	K:	1.00	X(1) = 6	X(2) = 8
Dim: 2	Var Red:	-0.385	FracSS:	0.494	K:	1.00	X(1) = 7	X(2) = 8

Ordered results for 2 dimensional models:

Total number of combinations:	45 (*)
Number tested	: 35 (*)
survivenum(2)	: 5 (*)
Number of survivors	: 5

Var Red:	6.172	FracSS:	0.553	X:	5	9
Var Red:	5.644	FracSS:	0.552	X:	2	5
Var Red:	5.267	FracSS:	0.543	X:	2	9
Var Red:	3.270	FracSS:	0.552	X:	3	5
Var Red:	3.184	FracSS:	0.550	X:	4	9

Dim: 3	Var Red:	6.121	FracSS:	0.555	K:	1.00	X(1) = 1	X(2) = 5	X(3) = 9
Dim: 3	Var Red:	8.613	FracSS:	0.578	K:	1.00	X(1) = 2	X(2) = 5	X(3) = 9
Dim: 3	Var Red:	6.256	FracSS:	0.551	K:	1.00	X(1) = 3	X(2) = 5	X(3) = 9
Dim: 3	Var Red:	6.217	FracSS:	0.546	K:	1.00	X(1) = 4	X(2) = 5	X(3) = 9

Dino Monico

```

Dim: 3   Var Red:   6.037   FracSS: 0.550   K: 1.00
X(1) = 5   X(2) = 6   X(3) = 9
Dim: 3   Var Red:   6.149   FracSS: 0.551   K: 1.00
X(1) = 5   X(2) = 7   X(3) = 9
Dim: 3   Var Red:   6.087   FracSS: 0.557   K: 1.00
X(1) = 5   X(2) = 8   X(3) = 9
Dim: 3   Var Red:   6.287   FracSS: 0.553   K: 1.00
X(1) = 5   X(2) = 9   X(3) = 10
Dim: 3   Var Red:   5.648   FracSS: 0.553   K: 1.00
X(1) = 1   X(2) = 2   X(3) = 5
Dim: 3   Var Red:   5.595   FracSS: 0.553   K: 1.00
X(1) = 2   X(2) = 3   X(3) = 5
Dim: 3   Var Red:   5.517   FracSS: 0.552   K: 1.00
X(1) = 2   X(2) = 4   X(3) = 5
Dim: 3   Var Red:   5.554   FracSS: 0.554   K: 1.00
X(1) = 2   X(2) = 5   X(3) = 6
Dim: 3   Var Red:   5.591   FracSS: 0.555   K: 1.00
X(1) = 2   X(2) = 5   X(3) = 7
Dim: 3   Var Red:   5.562   FracSS: 0.556   K: 1.00
X(1) = 2   X(2) = 5   X(3) = 8
Dim: 3   Var Red:   5.585   FracSS: 0.549   K: 1.00
X(1) = 2   X(2) = 5   X(3) = 10
Dim: 3   Var Red:   5.210   FracSS: 0.544   K: 1.00
X(1) = 1   X(2) = 2   X(3) = 9
Dim: 3   Var Red:   5.450   FracSS: 0.547   K: 1.00
X(1) = 2   X(2) = 3   X(3) = 9
Dim: 3   Var Red:   5.380   FracSS: 0.555   K: 1.00
X(1) = 2   X(2) = 4   X(3) = 9
Dim: 3   Var Red:   5.141   FracSS: 0.542   K: 1.00
X(1) = 2   X(2) = 6   X(3) = 9
Dim: 3   Var Red:   5.428   FracSS: 0.545   K: 1.00
X(1) = 2   X(2) = 7   X(3) = 9
Dim: 3   Var Red:   5.311   FracSS: 0.540   K: 1.00
X(1) = 2   X(2) = 8   X(3) = 9
Dim: 3   Var Red:   5.456   FracSS: 0.543   K: 1.00
X(1) = 2   X(2) = 9   X(3) = 10
Dim: 3   Var Red:   3.121   FracSS: 0.552   K: 1.00
X(1) = 1   X(2) = 3   X(3) = 5
Dim: 3   Var Red:   3.044   FracSS: 0.552   K: 1.00
X(1) = 3   X(2) = 4   X(3) = 5
Dim: 3   Var Red:   2.989   FracSS: 0.552   K: 1.00
X(1) = 3   X(2) = 5   X(3) = 6
Dim: 3   Var Red:   3.147   FracSS: 0.552   K: 1.00
X(1) = 3   X(2) = 5   X(3) = 7

```

Ordered results for 3 dimensional models:

```

Total number of combinations: 120 (*)
Number tested                  : 34 (*)
survivenum(3)                  : 5 (*)
Number of survivors            : 5

```

```

Var Red: 8.613   FracSS: 0.578   X: 2 5 9
Var Red: 6.287   FracSS: 0.553   X: 5 9 10
Var Red: 6.256   FracSS: 0.551   X: 3 5 9
Var Red: 6.217   FracSS: 0.546   X: 4 5 9
Var Red: 6.149   FracSS: 0.551   X: 5 7 9

```

Best Model Report Fold 1 (Test Set Data):

```

Model: 1   Var Red: 8.613   FracSS: 0.578   X: 2 5 9

```

Model: 2 Var Red: 6.287 FracSS: 0.553 X: 5 9 10

Timing Report:

dim: 1	Num_spaces: 10	Time: 1	Time/space: 0.10
dim: 2	Num_spaces: 35	Time: 4	Time/space: 0.11
dim: 3	Num_spaces: 34	Time: 4	Time/space: 0.12
Total:	Num_spaces: 79	Time: 9	Time/space: 0.11

Si illustrano ora i risultati della *kernel regression* ad alta *performance* (*KR*) tenendo fissi i parametri $H=4$ e $ORDER=0$ e facendo variare il numero di celle adiacenti (*NUMCELL*) in cui effettuare la ricerca dei *nearest neighbor* il cui numero (*NUMNN*) viene fatto a sua volta variare.

Osservando la tabella, riportata alla pagina successiva, si può notare che rispetto alla *FKR* con la *KR* si possono ottenere degli incrementi di *VR* considerevoli, ma anche fortissimi aumenti nella complessità computazionale evidenziati dal notevole tempo di elaborazione necessario (*Time*). Infatti, mentre il risultato migliore della *FKR* era pari a:

$$VR=8,613 \text{ e } Time=7 \text{ sec.};$$

ora si vedono molti valori di *VR* superiori (valori in grassetto) la cui media è pari a 8,980, ma il tempo medio di calcolo richiesto per ottenerli è pari circa a 1.550 secondi, cioè circa 220 volte il tempo richiesto dalla *FKR*.

Rispetto alla *FKR*, con la *KR* si ottiene un aumento medio di *VR* del 4,26%, ma anche un incremento medio del tempo del 22042%. Con questo non si vuole dire che la *KR* sia una metodologia da scartare, dipende molto dall'operatività che si intende seguire nel *trading* mobiliare: l'utilizzo della *KR* può essere ottimo se si opera *daily*, ma rischioso se non pessimo quando l'operatività è *intraday*, in particolar modo quando l'affluenza dei dati è ad intervalli inferiori ai 30 minuti.

h=4, ORDER=0, Leaf Cells= 16, Point Leaf=625				
SEARCH		ORDER 0		
NumNN	NumCells	VR	Time (sec.)	Time (Min.)
10	1	0,583	262	4
	2	0,921	562	9
	4	0,962	969	16
	8	1,017	1334	22
	16 (Tutte)	0,716	1424	24
25	1	6,565	269	4
	2	6,865	594	10
	4	6,811	1077	18
	8	6,968	1570	26
	16 (Tutte)	6,839	1464	24
50	1	7,904	298	5
	2	8,239	537	9
	4	8,325	981	16
	8	8,803	1624	27
	16 (Tutte)	8,751	1582	26
75	1	8,353	327	5
	2	8,678	562	9
	4	8,789	1012	17
	8	9,111	1472	25
	16 (Tutte)	9,051	1494	25
100	1	8,846	367	6
	2	8,605	615	10
	4	8,773	1135	19
	8	9,249	1598	27
	16 (Tutte)	9,153	1529	25
150	1	8,566	428	7
	2	8,697	701	12
	4	8,893	1146	19
	8	9,297	1696	28
	16 (Tutte)	9,336	1596	27
300	1	8,515	649	11
	2	8,593	943	16
	4	8,661	1437	24
	8	9,284	2001	33
	16 (Tutte)	9,346	1975	33
600	1	8,597	924	15
	2	7,901	1505	25
	4	8,059	2788	46
	8	8,850	3230	54
	16 (Tutte)	9,055	3283	55

4.4 UN APPROCCIO CON LE RETI NEURALI ARTIFICIALI

Ora si andranno a verificare i risultati che si riescono ad ottenere con le reti neurali (*NN*) per confrontare le due applicazioni illustrate precedentemente (*KR* e *FKR*) con una metodologia ben affermata nel mondo finanziario (*NN*) e trarre alcune conclusioni.

In questa sezione si andrà ad illustrare i risultati che si sono ottenuti applicando una semplice rete neurale artificiale (*NN*) a propagazione unidirezionale (*feed-forward*) con supervisore (*y*) per trattare la stessa matrice dati del cap. 4.1.

L'architettura della rete è la seguente:

- uno strato di input con:
 - 10 nodi: 10 *candidate predictors* (tabella A);
 - 3 nodi: 3 *best predictors* (tabella B);
- uno strato nascosto (*layer one*) con:
 - *n* nodi, dove *n* va da 5 a 15 (tabella A);
 - *n* nodi, dove *n* va da 1 a 11 (tabella B);
- uno strato di output con un unico nodo;
- funzione d'attivazione tra *input* e *layer one* sigmoide ('*tansig*');
- funzione d'attivazione tra *layer one* e output lineare ('*purelin*');
- *back propagation network training function* ('*trainlm*');
- *back propagation weight/bias learning function* ('*learngdm*');
- *performance function* ('*mse*');

in linguaggio Matlab:

```
net=newff(minmax(DATA(:,[X y])),[n,1], . . .  
         {'tansig','purelin'},'trainlm','learngdm','mse');
```

I parametri di training sono i seguenti:

```
net.trainParam.epochs=100;    % Maximum number of epochs to train
net.trainParam.goal=0;        % Performance goal
net.trainParam.max_fail=5;    % Maximum validation failures
net.trainParam.mem_reduc=1;   % To use for memory/speed tradeoff
net.trainParam.min_grad=1e-10; % Minimum performance gradient
net.trainParam.mu=0.001;     % Initial Mu
net.trainParam.mu_dec=0.1;   % Mu decrease factor
net.trainParam.mu_inc=10;    % Mu increase factor
net.trainParam.mu_max=1e10;  % Maximum Mu
net.trainParam.show=10;     % Epochs between showing progress
net.trainParam.time=inf;     % Maximum time to train in seconds
```

Dopo aver addestrato la rete e misurato il tempo di addestramento:

```
tic;
net=train(net,X.lrn,y.lrn);
toc;
```

con la simulazione della stessa:

```
ycalc.tst=sim(net,X.tst);
ycalc.evl=sim(net,X.evl);
```

si sono calcolate alcune misure di performance, tra le quali *VR*.

I risultati sono riportati di seguito nelle tabelle A e B.

A) 10 Candidate Predictors (X)				
NODI	VR (TST)	VR (EVL)	Time (sec.)	Time (min.)
5	5.15	7.08	329	5
6	4.86	6.68	411	7
7	3.76	5.53	501	8
8	5.65	7.10	600	10
9	5.22	7.33	710	12
10	5.02	5.70	837	14
11	5.62	6.91	935	16
12	5.31	7.22	1112	19
13	4.85	6.62	1336	22
14	5.07	5.53	1384	23
15	5.34	7.46	1515	25

B) 3 Best Predictors from FKR (x2, x5, x9)				
NODI	VR (TST)	VR (EVL)	Time (sec.)	Time (min.)
1	2.87	4.48	14	0
2	3.01	4.67	50	1
3	4.64	6.19	102	2
4	4.36	6.31	122	2
5	4.96	7.36	149	2
6	4.06	6.37	178	3
7	4.66	6.23	211	4
8	4.71	7.03	246	4
9	4.94	6.51	279	5
10	6.00	8.87	312	5
11	6.01	8.42	348	6

Come si può notare non ci sono grosse differenze tra le performance della rete neurale applicata a tutti i *candidate predictors* (tab. A) e quella applicata solo ai tre *best predictors* (tab. B), dove i tre *predictors* sono definiti *best* per costruzione (vedi *M-script* cap. 4.1), ma anche dalla *kernel regression*. Tuttavia, si possono osservare delle importanti riduzioni nel tempo computazionale (*Time*), quando si riducono i nodi di input della rete.

4.5 CONSIDERAZIONI FINALI

La *kernel regression* è operativamente simile alle reti neurali, tuttavia, in dettaglio le due metodologie sono totalmente differenti, ma abbastanza complementari. Un approccio che combina le migliori caratteristiche della *kernel regression* con quelle delle reti neurali può aiutare l'analista a sviluppare un buon modello.

Le due tecnologie vengono applicate entrambe nei problemi in cui non si conosce né il modello matematico sottostante né tanto meno se è possibile sviluppare un modello robusto utilizzando i *candidate predictors*. Lo scopo del processo di modellamento è quello di riuscire a predire una variabile di output basata su un set di *candidate predictors*.

Il tipico approccio con *KR* e *NN* è basato su due paradigmi: *training* e *testing*. Ogni modello è basato sul *training set* e successivamente è testato con l'utilizzo del *testing set*. Mentre le reti neurali modellano con processi iterativi, nei quali i pesi associati tra gli input e gli output di un neurone sono modificati fino a che non viene raggiunto un certo livello di performance, il metodo della *kernel regression* non è iterativo.

Dal punto di vista computazionale quanto detto implica che il processo di *training* con le reti neurali è molto lungo, ma una volta che il sistema converge ad un modello, il processo di *testing* è molto rapido. Viceversa con *KR* il processo di *learning* è estremamente rapido ed il tempo di *testing* può essere altrettanto rapido se si sacrifica un po' di performance (*FKR*), tuttavia utilizzando tutte le potenzialità di *KR* può diventare molto lento (vedi cap. 3.4).

La natura complementare delle due tecnologie può essere riassunta nella tabella seguente, in cui si mostra le relazioni tra la qualità del modello, il

tempo di *training* e quello di *testing* rispettivamente nelle reti neurali (*NN*) e nella *kernel regression* (*KR*).

	Model quality	Preparation time	Testing time
NN	Low	Slow	Fast
NN	High	Slower	Fast
KR	Low	Fast	Fast
KR	High	Fast	Slow

La qualità del modello è specificata dall'analista attraverso l'utilizzo dei parametri e i loro differenti valori. Con le reti neurali l'analista può alterare la qualità del modello cambiando l'architettura della rete (il numero di neuroni, il numero di strati nascosti, le funzioni di attivazione, la funzione d'apprendimento...). Allo stesso modo ci sono dei parametri della *kernel regression* che possono alterare la qualità del modello: l'ordine dell'algoritmo, la dimensione massima del polinomio, l'altezza massima del *p-Tree* o il numero di punti per cella, il modo di ricerca dei *nearest neighbors* (si veda il cap. 4.2).

Per quanto riguarda la bontà dei modelli individuati dalle due metodologie, si può dire che con la matrice dati proposta (cap. 4.1) la *KR* ha portato a risultati migliori rispetto alla *NN*, ma la relazione tra performance e valori assunti dai parametri della *kernel regression* è stata più visibile, ciò comporta per l'analista un maggior controllo sui parametri.

In generale, senza bocciare l'approccio alle reti neurali, per problemi ad alta dimensionalità con serie storiche molto lunghe, la strategia di modellamento diventa:

1. applicare una veloce tecnica di ricerca come la *FKR* per identificare gli spazi più promettenti;
2. utilizzare la *KR* oppure le *NN* per esaminare gli spazi più promettenti in dettaglio.

Questo tipo di approccio dovrebbe dare al *data miner* una probabilità maggiore di successo nel modellare serie storiche come quelle dei mercati finanziari, in particolare il *Fib30* o il *MiniFib30*.

APPENDICE E-MAIL

-----Messaggio originale-----
Da: Dino [<mailto:dino.monico@libero.it>]
Inviato: giovedì 26 luglio 2001 19.09
A: Wolberg@hitech.technion.ac.il
Oggetto: Question?

Dear Dr. Wolberg,

regarding your „Expert Trading System“ book, which I had the pleasure to read, I will very grateful if you answer the following similar questions.

Having calculated the \hat{y}_j values in every single test point with a polynomial (order, dimensions, kernel, the three options), how could I make previsions for the future without knowing the future predictors values, but only the time?

How can I locate the future point in the p-Tree cells if I don't know the future predictors value?

How can I significantly transpose the past values in the future if I can't calculated the new polynomial coefficients?

Thank you in advance.

Sincerely,

Dino.

-----Messaggio originale-----

Da: John Wolberg [<mailto:jwolber@attglobal.net>]

Inviato: giovedì 26 luglio 2001 22.48

A: dino.monico@libero.it

Oggetto: Re: Question?

Dear Dino,

The X values (i.e., the predictors are used to compute \hat{y} for the test points) must be defined so that they can be computed before the actual Y values are know. So once you start using the system for making actual predictions, the same p-Tree is used in the same way that you used it to compute the \hat{y} values for the test points.

For time series work (like making stock market predictions) the X's are "backward looking" and the Y is a forward looking variable. For example, a typical X might be the fractional change over the last day and the Y might be the fraction change from today to tomorrow.

I hope this clears up your problem.

All the best...

John Wolberg.

-----Messaggio originale-----

Da: John Wolberg [<mailto:jwolber@hitech.technion.ac.il>]

Inviato: giovedì 6 dicembre 2001 10.28

A: dmonico@libero.it

Oggetto: Re: Some Questions!

Hi Dino,

here are some answers to your questions:

Why in your Book do you consider only the exponential function about kernel? Is it better than others function?

Our experience has shown that the choice of kernel isn't all that important for noisy problems like market modelling. In fact, most of our works uses fast mode which is just a 0 or 1 kernel (i.e., a point is either near enough to be used and thus gets a kernel of 1 or is excluded which is equivalent to getting a kernel value of 0).

In earlier work that I did with kernel regression, we wrote code which allowed exponential kernels (i.e., $\exp(-K * D^2)$ where D is distance) and Cauchy kernels (i.e., $1 / (1 + K * D^2)$ which also has a value of 1 when $D=0$ and 0 when D is infinite).

We never found any significant reason for choosing one over the other so chose the exponential just because it is aesthetically more appealing.

*Is the p-Tree a "Classification Tree" or a "Regression Tree"?
Which is the more appropriate name and Why?*

It is not a classification tree and I've never heard the word "Regression Tree". All it is a data structure that permits one to rapidly process the data by quickly discovering nearby neighbors.

Use of this tree avoids the need to look at all points to discover nearby points. For large data sets this becomes crucial.

I hope this answers your questions. All the best... John.

BIBLIOGRAFIA

- Wolberg R. John. *Expert Trading Systems: Modeling Financial Markets with Kernel Regression*. John Wiley & Son, 1999.
- Wolberg R. John. *Fast kernel regression documentation*. 22 Settembre 1999.
- Wolberg R. John. *FKR Help*. E-mails to Dino Monico. Da Aprile a Dicembre 2001.
- Kimche Ronen. *FKR Document Release 18*. E-mail to Dino Monico. 24/05/2001.
- Goutte Cyril. *Kernel regression files*. E-mails to Dino Monico. 10 e 23/10/2001.
- Torgo Luis. *Kernel regression trees*. Porto: LIACC University, 2000.
- Torgo Luis. *KRT - full paper question*. E-mail to Dino Monico. 15/11/2001.
- Torgo Luis. *Functional models for regression tree leaves*. Porto: LIACC, 2000.
- Hoti Fabian. *Kernel regression via binned data*. Helsinki: Rolf Nevanlinna Institute, University of Helsinki. (Cap.2-3). 01 Febbraio 2001.
- Hardle W. *Applied non parametric regression*. Cambridge University Press. (Cap.1-2-3). 1990.
- Wand M.P., Jones M.C. *Kernel Smoothing*. Chapman & Hall. (Cap.5). 1994.
- Demuth H., Beale M. *Neural Networks Toolbox for use with Matlab*. The MathsWorks Inc., Natick. 1997.
- Azoff E.M. *Neural Network time series forecasting of Financial Markets*. JohnWiley & Son. 1994.
- Refenes A.P. *Neural Network in the Capital Markets*. John Wiley & Son. 1995.
- Di Lorenzo Roberto. *Come guadagnare in borsa con l'analisi tecnica 1, 2, 3*. Il Sole 24 ore. 2000.
- Borsa Italiana SpA. *Il Fib 30*. <http://www.borsaitaliana.it>.