Applicazione di modelli di regressione sui prezzi di chiusura settimanale del Bitcoin

Alessandro Resta

10 Marzo 2022

1 Introduzione

Il Bitcoin è una moneta virtuale gestita interamente da un network di nodi che, attraverso un database condiviso, registra e mantiene le transazioni che vengono effettuate tra i vari utilizzatori del sistema. Viene classificata dagli esperti di finanza come "riserva di valore" attualmente volatile, questo perché il suo valore è determinato esclusivamente dalla leva domanda e offerta. Per questo motivo un problema molto interessante consiste nello stimare il prezzo che questa moneta (o criptovaluta) assumerà in futuro, in modo da poterne identificare, per esempio, il suo andamento.

Il valore del bitcoin viene spesso espresso in dollari, dunque una stima del suo prezzo dovrà riguardare valori continui. Gli strumenti più adatti a questo tipo di analisi sono i modelli di regressione, che permettono di stimare una relazione funzionale tra la variabile dipendente (detta anche risposta o target) e le variabili indipendenti (dette anche features o predittori). Più avanti verranno chiariti questi concetti.

Lo scopo di questa trattazione consiste nel testare diversi algoritmi di regressione, al fine di individuare quello che meglio stima il **prezzo di chiusura settimanale** (prezzo del bitcoin alle ore 23:59 di ogni domenica), partendo da altri dati e commettendo il minimo errore nelle predizioni.

Il linguaggio di scripting scelto è **R**, in quanto molto adatto alla creazione di soluzioni algoritmiche per problemi statistici, come in questo caso. I modelli di regressione saranno implementati mediante il package **caret**, che mette a disposizione una serie di metodi per il training e il testing degli algoritmi. Fornisce anche la possibilità di impostare dei parametri per definire un modello di validazione, al fine di fare tuning dei parametri dei regressori e individuare quello ottimale.

Data la numerosità di modelli di regressione esistenti, si è deciso di selezionare i quattro più utilizzati, ovvero: **Multiple Linear Regressor**, **Decision Tree Regressor**, **Random Forest Regressor** e **Support Vector Regressor**.

Nel seguito i modelli sopra citati saranno approfonditi, discutendo i risultati prodotti per ognuno e, alla fine, si metteranno a confronto per decretare il modello migliore per il dataset utilizzato.

Le misure statistiche per la valutazione delle performance degli algoritmi di regressione utilizzate sono il Root Mean Squared Error (**RMSE**), il coefficiente di determinazione (**R-Squared**) e il Mean Absolute Error (**MAE**). Tutte e tre le misure saranno spiegate nelle sezioni successive.

Il dataset è preso dal sito: `https://www.kaggle.com/varpit94/bitcoin-data-updated-till-26jun2021` e contiene lo storico giornaliero in dollari dei seguenti dati relativi al bitcoin: data, prezzo massimo, prezzo minimo, volume di scambio, prezzo di apertura, prezzo di chiusura e prezzo di chiusura rettificato.

La data (**Date**) indica giorno, mese e anno dell'osservazione dei valori dei restanti attributi; il prezzo massimo e minimo (**High e Low**) è il prezzo in dollari più alto e più basso rispettivamente che il bitcoin ha raggiunto in un trading day; il volume di scambio (**Volume**) corrisponde al controvalore in dollari di tutte le transazioni effettuate in un trading day; il prezzo di apertura e chiusura (**Open e Close**) è il prezzo in dollari del bitcoin alle ore 00:00 e alle ore 23:59 rispettivamente di uno stesso trading day. Il prezzo di chiusura rettificato (**Adj Close**) non presenta differenze di valore col prezzo di chiusura, pertanto saranno considerati uguali.

Come anticipato sopra, siamo interessati al prezzo di chiusura settimanale del bitcoin. Questa scelta è dovuta al fatto che il bitcoin è molto volatile, di conseguenza per semplificare lo studio dell'andamento dei prezzi si preferisce avere una finestra temporale più ampia rispetto a quella giornaliera, in modo che le variazioni di prezzo siano più contenute.

2 Preprocessing del Dataset

La fase di preprocessing del dataset mira alla rimozione di attributi non necessari, alla costruzione di un nuovo dataset che presenti i valori degli attributi su scala settimanale anziché giornaliera e infine alla normalizzazione e suddivisione di quest'ultimo in training set e test set.

Il dataset di partenza è mostrato in figura 1.

Date	Open	High	Low	Close	Adj.Close	Volume
2014-09-22	399.100	406.916	397.130	402.152	402.152	24127600
2014-09-23	402.092	441.557	396.197	435.791	435.791	45099500
2014-09-24	435.751	436.112	421.132	423.205	423.205	30627700
2014-09-25	423.156	423.520	409.468	411.574	411.574	26814400
2014-09-26	411.429	414.938	400.009	404.425	404.425	21460800
2014-09-27	403.556	406.623	397.372	399.520	399.520	15029300

Figure 1: Dataset iniziale

2.1 Rimozione di attributi non necessari

Un attributo che si reputa non necessario, e che pertanto non verrà approfondito, è il prezzo di chiusura rettificato, in quanto coincide per valore al prezzo di chiusura. Per questo motivo verrà rimosso. Anche gli attributi **Week** e **Year** (descritti in seguito e rappresentati nella figura 2) verranno rimossi, ma solo alla fine della fase di preprocessing, perché utilizzati solo come supporto alla crezione di un nuovo dataset di osservazioni settimanali.

2.2 Creazione del dataset di osservazioni settimanali

Al fine di un corretto passaggio dei valori degli attributi da giornalieri a settimanali si introduce una nuova colonna chiamata **Week**, che contiene, per ciascuna osservazione, il numero di settimana rispettivo. Un'altra modifica riguarda il campo **Date** che viene sostituito da **Year**, ovvero l'anno corrispondente a tale data. L'idea è avere una enumerazione delle settimane, da 1 a 52 (o 53), affiancata al relativo anno. Per esempio:

$$28/12/2014 \rightarrow (52, 2014)$$

Ovvero il 28 dicembre del 2014 è nella 52-esima settimana del 2014. In questo modo è possibile raggruppare le tuple (o osservazioni) del dataset per settimana e anno, attraverso la funzione **aggragate()** di R. Nel raggruppamento si applicano delle funzioni specifiche su ciasun attributo, che restituiscono i valori assunti da quest'ultimi dopo l'aggregazione. In particolare:

- All'attributo **Hight** si applica la funzione **max**, che trova il massimo prezzo del bitcoin settimanale
- All'attributo **Low** si applica la funzione **min**, che trova il minimo prezzo del bitcoin settimanale
- All'attributo **Volume** si applica la funzione **mean**, che trova la media dei volumi di scambio del bitcoin settimanale.

- All'attributo **Open** si applica la funzione **head** con parametro **1**, in modo da individuare il prezzo di apertura del bitcoin del primo giorno della settimana
- All'attributo **Close** si applica la funzione **tail** con parametro **1**, in modo da individuare il prezzo di chiusura del bitcoin dell'ultimo giorno della settimana

Di seguito è riportata una rappresentazione del dataset dopo l'aggiunta della colonna Week e la modifica della colonna Date in Year.

Week	Year	High	Low	Volume	Open	Close
39	2014	441.557	374.332	26681800	399.100	377.181
40	2014	391.379	289.296	39522557	376.928	320.510
41	2014	382.726	302.560	48736115	320.389	378.549
42	2014	411.698	368.897	22414581	377.921	389.546
43	2014	392.646	342.877	16241686	389.231	354.704
44	2014	359.984	320.626	15296529	354.777	325.892

Figure 2: Dataset dopo la prima modifica

2.2.1 Problema del raggruppamento per settimana e anno

La numerazione delle settimane è effettuata mediante la funzione di R **strftime()** con parametro **format="%V"**. Nel caso in cui una settimana condivida gli ultimi giorni del mese di dicembre di un anno con i primi giorni del mese di gennaio dell'anno successivo, questa funzione fa il reset del conteggio delle settimane solo se ci sono più giorni di gennaio che di dicembre. Per esempio:

$$29/12/2014 \rightarrow 1$$

$$01/01/2015 \rightarrow 1$$

Questo accade perché il 29 dicembre del 2014 e il 1 gennaio 2015 sono lunedì e giovedì della stessa settimana, che contiene un numero maggiore di giorni di gennaio rispetto a dicembre.

Analogamente è possibile che nella settimana a cavallo dei due anni ci siano più giorni di dicembre che di gennaio, e dunque l'enumerazione non ripartirà da 1, ma proseguirà con il conteggio. Per esempio:

$$31/12/2016 \rightarrow 52$$

$$01/01/2017 \rightarrow 52$$

In questo caso il 31 dicembre 2016 e il 1 gennaio 2017 sono sabato e domenica della medesima settimana. Quindi nella stessa settimana ci sono più giorni di dicembre e dunque non si avrà un reset del conteggio delle settimane.

Secondo la modifica degli attributi descritta sopra il primo esempio diventerà:

$$29/12/2014 \rightarrow (1, 2014)$$

$$01/01/2015 \rightarrow (1, 2015)$$

Mentre il secondo esempio sarà:

$$31/12/2016 \rightarrow (52, 2016)$$

$$01/01/2017 \rightarrow (52, 2017)$$

Come si potrà intuire facilmente in entrambi gli esempi, un eventuale raggruppamento per settimana e anno tratterà le coppie di osservazioni come non appartenenti alla stessa settimana dello stesso anno (anche se così è), portando alla creazione di gruppi differenti, e dunque ad un'errata aggregazione dei giorni in settimane.

2.2.2 Possibile soluzione al problema del raggruppamento

Una possibile soluzione potrebbe essere quella di far rientrare i giorni di dicembre per cui avviene il reset del conteggio delle settimane nell'ultima settimana completa di dicembre. Per esempio:

$$28/12/2014 \rightarrow (52, 2014)$$

$$29/12/2014 \rightarrow (1, 2014) \Longrightarrow 29/12/2014 \rightarrow (52, 2014)$$

$$30/12/2014 \rightarrow (1, 2014) \Longrightarrow 30/12/2014 \rightarrow (52, 2014)$$

$$31/12/2014 \rightarrow (1, 2014) \Longrightarrow 31/12/2014 \rightarrow (52, 2014)$$

In questo modo è come se l'ultima settimana di dicembre 2014 (la 52-esima) venga estesa con i giorni di dicembre che hanno subito il reset del conteggio delle settimane. Analoga soluzione può essere applicata al problema speculare, ovvero si estende la prima settimana di gennaio con i giorni di gennaio per cui non è scattato il reset delle settimane. Per esempio:

$$02/01/2017 \rightarrow (1, 2017)$$

$$01/01/2017 \rightarrow (52, 2017) \Longrightarrow 01/01/2017 \rightarrow (1, 2017)$$

Dove il 2 gennaio 2017 è il primo giorno della prima settimana del 2017, mentre il 1 gennaio 2017 sarebbe l'ultimo giorno dell'ultima settimana di dicembre dell'anno prima. Dunque si estende la prima settimana di gennaio 2017 con quest'ultimo giorno.

Dopo aver applicato le suddette modifiche a tutto il dataset, il raggruppamento per settimana e anno non crea più nessun tipo di problema. Ovviamente nel calcolo dei nuovi valori degli attributi si deve tenere conto che alcuni raggruppamenti potrebbero riguardare sequenze di giorni più lunghe di sette; ad esempio nel calcolo della media settimanale dei volumi di scambio potrebbero essere considerati otto giorni, portando dunque il risultato ad essere diverso rispetto quello previsto con sette giorni. Trattandosi di casi particolari che si manifestano solo nella settimana finale di ciascun anno si ritiene irrilevante l'errore commesso nel computo.

2.3 Normalizzazione e suddivisione del nuovo dataset in training set e test set

Successivamente alla crezione del detaset di osservazioni settimanali, si procede alla sua normalizzazione, che consiste nello scalare i valori degli attributi features in modo da farli rientrare nel range [0,1]. Questo è importante perché il bitcoin negli anni ha subito grandi variazioni di prezzo e dunque si è in presenza di valori in scale diverse tra loro, che possono portare a predizioni meno accurate. La formula di normalizzazione utilizzata è:

$$x = \frac{x - min(X)}{max(X) - min(X)}$$

Con $\mathbf{X}$ vettore di valori di un generico attributo del dataset e $\mathbf{x} \in \mathbf{X}$

Infine si suddivide il dataset così ottenuto in training set e test set rispettando le percentuali **75% training** e **25%test**. Prima della suddivisione è stato effettuato uno **shuffle** del dataset, in modo da rendere ancora più random il partizionamento.

Proprio per un fattore randomico, si è pensato di impostare, tramite la funzione di R **set.seed()**, un seed di **689** per lo shuffling e **789** per il sampling usato nella suddivisione, in modo da poter replicare i futuri risultati ottenuti.

Per lo shuffling è stata utilizzata la funzione **sample()** di R, che passando un solo parametro, ovvero un array di valori **x**, restituisce una permutazione di **x**. Dunque passando il vettore **x=[1...N]** con N dimensione del dataset viene ritornata una permutazione casuale di N indici che possono essere usati per costruire un dataset shuffled.

Per la suddivisione si fatto uso della funzione **createDataPartition()** di caret, i cui parametri principali sono il vettore dell'attributo target (in questo caso Close) e la percentuale di tuple del dataset da considerare (quindi 0.75). Il risultato è composto dall'insieme di indici delle tuple selezionate, che possono essere assegnate ora ad un dataset di training; i rimanenti possono invece essere utilizzati per la creazione di un dataset di test.

Le tuple appartenenti al training set saranno utilizzate per allenare i modelli di regressione, mentre le tuple del test set saranno utilizzate per testarne il comportamento e dunque le performance.

Le percentuali scelte sono in linea con quelle convenzionalmente utilizzate. Questa partizione è di fondamentale importanza per l'applicazione di tecniche di regressione, poiché queste prevedono due macro fasi: **learning** e **test**.

Nella fase di learning si utilizzano le osservazioni del training set per trovare i parametri ottimali di funzioni che, congiuntamente alle features, permettono di ottenere un valore di stima per la variabile dipendente (che nel caso qui analizzato consiste nel prezzo di chiusura settimanale del bitcoin).

Trovata la funzione di regressione, si procede a valutarne la bontà mediante applicazioni su tuple del test set, di cui conosciamo il valore di output reale e dunque possiamo estrapolare l'accuratezza delle predizioni.

2.4 Dataset dopo la fase di preprocessing

Al termine delle modifiche applicate al dataset di partenza si arriva alla conclusione della fase di preprocessing, dove il dataset risultante contiene solamente gli attributi features e l'attributo di output. Questi sono rispettivamente High, Low, Volume e Close e sono mostrati in figura 3.

High	Low	Volume	Open	Close
0.617142	0.56490419	0.30977396	0.53854935	39974.894531
0.10829477	0.11258039	0.20869607	0.11173559	7193.599121
0.00034434	0.00092519	9.653e−05	0.00035759	235.977005
0.10973463	0.11130796	0.04615812	0.10962883	7720.25
0.08698	0.0801116	0.01894723	0.08384802	6008.419922
0.01529506	0.01595611	0.00191275	0.01458141	1267.119995

Figure 3: Dataset dopo il preprocessing

3 Implementazione dei Modelli di Regressione con Caret

Come anticipato ad inizio trattazione e visto concretamente nel capitolo dedicato al preprocessing del dataset, i dati a disposizione sono continui e dunque la loro analisi deve essere supportata da strumenti adatti a trattare questa tipologia di valori. In questa sezione verranno discussi ed applicati i modelli di regressione menzionati ad inizio trattazione, vale a dire **Multiple Linear Regressor**, **Decision Tree Regressor**, **Random Forest Regressor** e **Support Vector Regressor**.

Per tutti questi regressori si darà una breve introduzione e si discuterà la loro implemenzazione in R attraverso i metodi disponibili all'interno di caret. Dato che quest'ultimo fornisce un tool di creazione di algoritmi di machine learning unificato, si farà uso di uno stesso insieme di metodi per tutti i regressori. Le differenze risiederanno nei parametri scelti per la creazione del modello, che cambieranno a secondo del regressore trattato.

In particolare i metodi utilizzati sono **trainControl()**, **train()** e **predict()**.

trainControl() è la funzione che permette di costruire un modello di validazione, che sarà utilizzato per fare tuning dei parametri del modello di regressione, ottimizzandolo. Il metodo di validazione è rappresentato da una stringa ben precisa, che va passata al parametro **method**.

train() è la funzione che crea il modello di regressione, passato al parametro **method**. Al parametro **form** va passata la formula da utilizzare, del tipo $\mathbf{Y} \sim \mathbf{X}_1 + \ldots + \mathbf{X}_n$, dove Y rappresenta la variabile target di cui si vuol predire il valore mentre $\mathrm{X}_1 \ldots \mathrm{X}_n$ rappresentano le variabili features usate per la predizione. Ai parametri **data** e **trControl** vanno passati il dataset di training e l'oggetto restituito da trainControl() rispettivamente.

predict() è la funzione che, passato un modello al parametro **model** e un dataset di test al parametro **data**, permette di utilizzare il modello e verificarne il comportamento su dati di test mai incontrati durante il training.

La lista di tutti i metodi di validazione e dei modelli di regressione disponibili si può trovare nella documentazione di caret [4].

Si noti che la creazione del modello può essere influenzata da fattori aleatori nel caso di utilizzo di certi modelli di validazione, pertanto il seed da impostare prima di effettuare il training è **111** se si vogliono ottenere i risultati presenti in questa trattazione.

Le misure di performance utilizzate per la valutazione della bontà dei regressori sono, come anticipato nell'introduzione, RMSE, R-Squared e MAE.

L'RMSE è una misura dell'errore medio commesso dal regressore nelle predizioni, e consistente nella radice quadrata dello scarto quadratico medio tra valore predetto dal regressore e valore reale. La sua formula matematica è:

$$RMSE = \sqrt{\frac{\sum (y_p - y_r)^2}{N}}$$

Dove y_p e y_r sono rispettivamente valore predetto e valore reale, mentre N è la dimensione del test set.

L'R-Squared, detto anche coefficiente di determinazione, misura quanto i valori predetti dal regressore sono vicini alla retta di regressione. Più questo valore è vicino a 1, più la retta di regressione approssima meglio i dati. La sua formula matematica è:

$$R - Squared = \frac{\sum (y_p - \mu_p)(y_r - \mu_r)}{\sigma_{y_p} \sigma_{y_r}}$$

Con μ_p e μ_r media delle predizioni e media dei valori reali rispettivamente e σ_{y_p} e σ_{y_r} deviazione standard delle predizioni e dei valori reali rispettivamente.

L'MAE è una misura di errore medio simile a RMSE, ma che prevede il valore assoluto della differenza tra predizioni e valori reali (anziché l'elevamento al quadrato) e non computa la radice quadrata. La sua formula matematica è:

$$MAE = \frac{\sum |y_p - y_r|}{N}$$

3.1 Multiple Linear Regression

Il Multiple Linear Regressor (**MLR**) è un algoritmo di regressione lineare che utilizza delle variabili indipendenti chiamate predittori per predire una variabile indipendente chiamata risposta. Questa tecnica di regressione cerca una funzione lineare di più variabili che meglio approssima i dati presenti su un certo dataset.

Data la natura lineare di questo regressore è bene applicarlo nei casi in cui delle correlazioni lineari sono possibili tra i predittori e la risposta. Nel caso oggetto di studio, i predittori corrispondono agli attributi del dataset del bitcoin High, Low, Volume e Open; mentre la risposta è rappresentata dall'attributo Close.

Una funzione che permette la visualizzazione grafica di eventuali correlazioni tra attributi è **ggpairs()** del package **GGally**. Questa funzione mostra la correlazione tra tutte le coppie di attributi presenti nel dataset passatogli come parametro. Le coppie di maggior interesse sono quelle che coinvolgono l'attributo Close, ovvero la variabile risposta. In figura 4 si può osservare il risultato del plot sul dataset di figura 3.

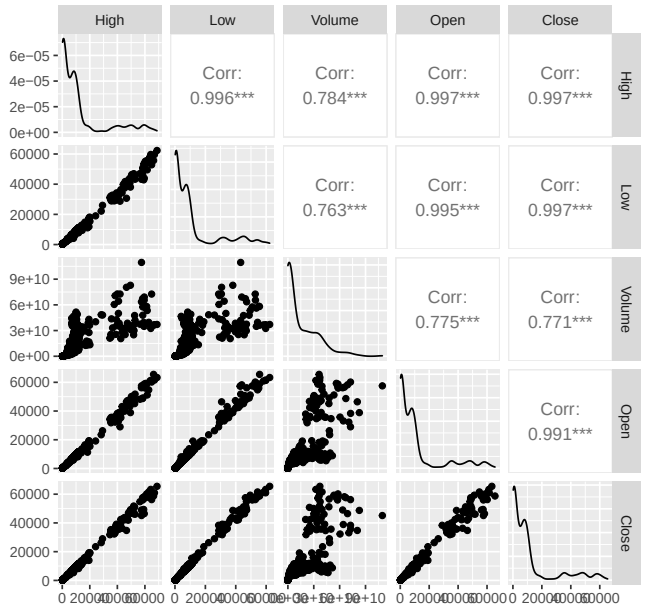

Figure 4: Correlazioni degli attributi del dataset di bitcoin

Osservando i riquadri dell'ultima riga della figura sopra, si può notare che la correlazione tra Close e tutti gli altri attributi è molto forte, ad eccezione del Volume, che presenta una correlazione più debole. Nella diagonale superiore si può avere la conferma di quanto appena asserito, poiché i valori di correlazione sono molto vicini ad 1. Si può concludere pertanto che il modello MLR può

potenzialmente dimostrarsi performante.

3.1.1 Applicazione di MLR sul dataset di bitcoin

Si applica ora il modello di regressione appena descritto al dataset di figura 3. Si è stabilito a priori un modello di validazione che verrà applicato a tutti gli algoritmi di regressione oggetto di analisi.

In particolare si è utilizzato un modello di tipo **repeated cross validation** con **10 fold** e **5 ripetizioni**. Il dataset di training verrà dunque suddiviso in 10 parti, ciascuna contenente uno stesso numero di tuple (una parte potrebbe contenerne più o meno di 10 se la divisione non è perfetta). Dopo questa operazione si procederà a reiterare per 5 volte il processo di selezione di una delle 10 fold, su cui verrà fatto il tuning dei parametri.

La formula da passare al metodo train() è **Close∼.**, in cui si può specificare solo l'attributo da predire. La stringa identificativa da passare a method è invece **lm**.

Passando il modello appena creato, insieme al dataset di test, al metodo predict() si ottengono dei valori su cui è possibile stimare l'errore con le formule viste alla fine della sezione 3. Il calcolo produce i seguenti risultati:

- **RMSE: 567.24**
- **R-Squared: 0.9990**
- **MAE: 281.22**

Per interpretare da un punto di vista grafico le misure ottenute sopra, si osservi il plot di figura 5, in cui nell'asse x e y sono presenti i valori predetti dell'attributo Close e i loro residui (differenza che questi hanno rispetto ai valori reali) rispettivamente.

La linea rossa orizzontale passa per i valori predetti dal regressore che non presentano differenze con i valori reali.

Come si può notare si ha una maggiore accuratezza delle predizioni per valori bassi di Close, mentre all'aumentare di questi la dispersione aumenta di conseguenza, portando ad un errore di stima maggiore.

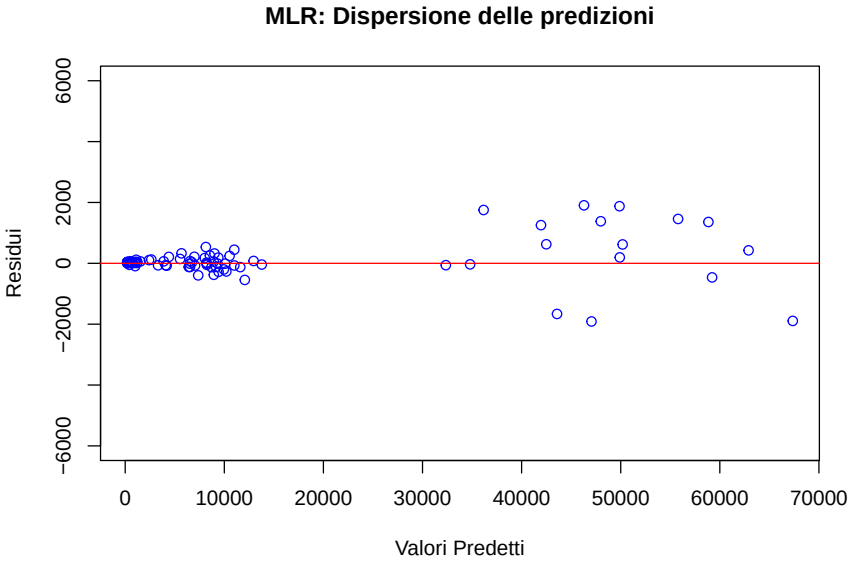

Figure 5: MLR: dispersione delle predizioni

3.2 Decision Tree Regressor

Il Decision Tree (**DT**) è un diagramma a forma di albero, dove ogni nodo interno contiene un test su un attributo, ogni ramo rappresenta un risultato del test, e ogni nodo foglia (o terminale) contiene un determinato valore (continuo o discreto). Nel caso in cui l'attributo da predire è a valori continui si parla di Decision Tree Regressor (**DTR**), e sarà proprio questo ad essere utilizzato per stimare il prezzo del bitcoin.

Il DT è uno strumento molto utilizzato per risolvere problemi di classificazione e regressione, il cui obiettivo consiste nel costruire una partizione ben precisa del dataset di training. Gli elementi della partizione sono associati, ciascuno, a foglie diverse dell'albero e, messi assieme, formano il dataset di partenza.

Questo processo di partizionamento è effettuato in modo ricorsivo tramite i test sugli attributi features rappresentati dai nodi interni, finché i nodi non diventano **puri** oppure non si può più procedere nella suddivisione per mancanza di attributi. Un nodo si dice puro se le tuple ad esso associato assumono tutte uno stesso valore nell'attributo da predire.

A seconda del tipo di dati di cui si dispone, il valore contenuto nelle foglie può essere una particolare classe oppure un valore continuo estratto, per esempio, dalla media dei valori reali associati all'attributo da stimare, relativa alle sole tuple finite nell'insieme del nodo foglia.

L'ordine con cui vengono testati gli attributi può influenzare l'albero finale risultante, pertanto nascono diversi algoritmi che, seguendo delle euristiche ben precise, stabiliscono una misura di **goodness**. Questa misura viene utilizzata per stabilire un ordine di scelta degli attributi da testare all'interno del DT.

Per questa trattazione si è deciso di adottare l'algoritmo **CART**, già presente nel package caret. CART è basato sul concetto di **gini index**, ovvero un valore associato ad ogni nodo dell'albero che misura l'impurità al suo interno. La formula del gini index è data da:

$$gini(S_x) = 1 - \sum_{i=1}^{N} p_i$$

Dove S_x è l'insieme contenente N tuple di un certo nodo X e p_i è la probabilità che una tupla dell'insieme S_x sia di classe i.

Una volta partizionato un nodo X in k sottoinsiemi $S_1,\ldots,S_k$ si stabilisce il **gini split** per X, dato da:

$$gini_{split}(S_x) = \sum_{i=1}^{k} \frac{|S_i|}{|S_x|} gini(S_i)$$

L'attributo che minimizza il gini split viene scelto per primo nella sequenza di attributi da testare.

3.2.1 Applicazione di DTR sul dataset di bitcoin

Applicando quanto appena detto al problema di stima del prezzo del bitcoin, si ottiene l'albero CART mostrato in figura 6, previa creazione del modello tramite la dicitura **rpart** nel metodo train() per scegliere il CART DTR.

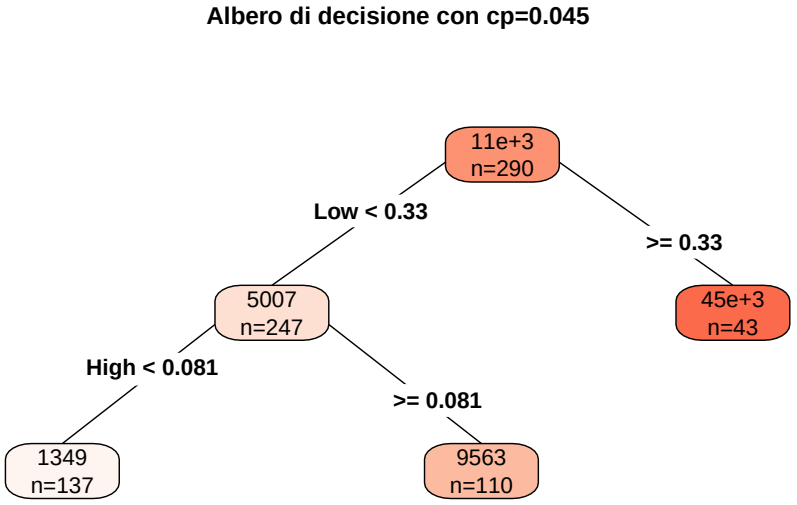

Figure 6: CART DT: tuning automatico su 3 valori

All'interno di ciascun nodo, il valore in alto indica la predizione dell'attributo target Close, mentre quello più in basso (chiamato **n**) indica il numero di tuple del training set finite nell'insieme associato a quel nodo.

L'albero prodotto risulta essere molto semplice, nonostante CART abbia riutilizzato più volte i due parametri con misura di goodness più elevata Low e High per rendere più selettivi i test e dunque ottenere più foglie.

La configurazione dell'albero ottenuta deriva dal parametro complexity parameter (o **cp**) del modello DTR, che è stato impostato in automatico a **cp=0.045** da caret durante la fase di tuning mostrata in figura 7.

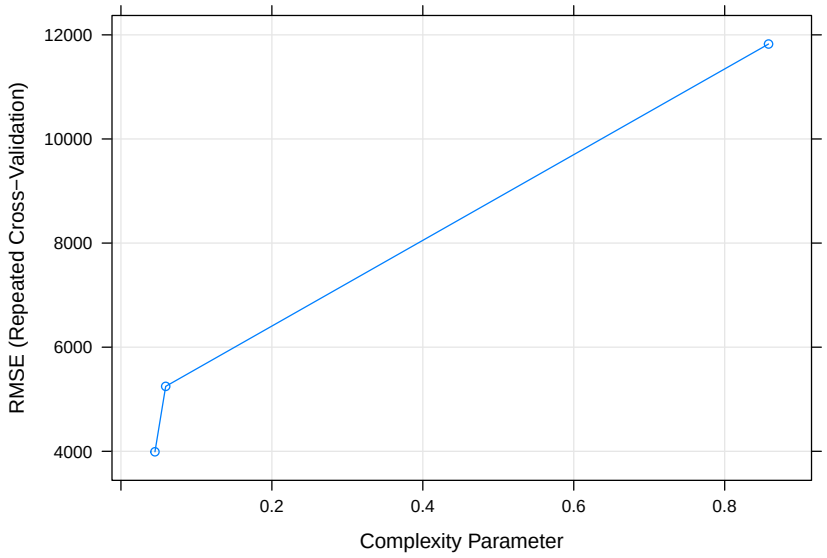

Figure 7: CART DTR: fase di tuning di cp su 3 valori

Andando a calcolare le misure di performance di questo modello si ottiene:

- **RMSE: 4522.4**
- **R-Squared: 0.94344**
- **MAE: 2546.5**

La performance di questo modello è abbastanza scarsa, specialmente se confrontata con il MLR.

Tuttavia aumentando il numero di valori su cui effettuare il tuning del parametro cp fino a **10** è possibile ottenere dei risultati nettamente superiori, a discapito di un albero di decisione più complesso come mostrato in figura 8.

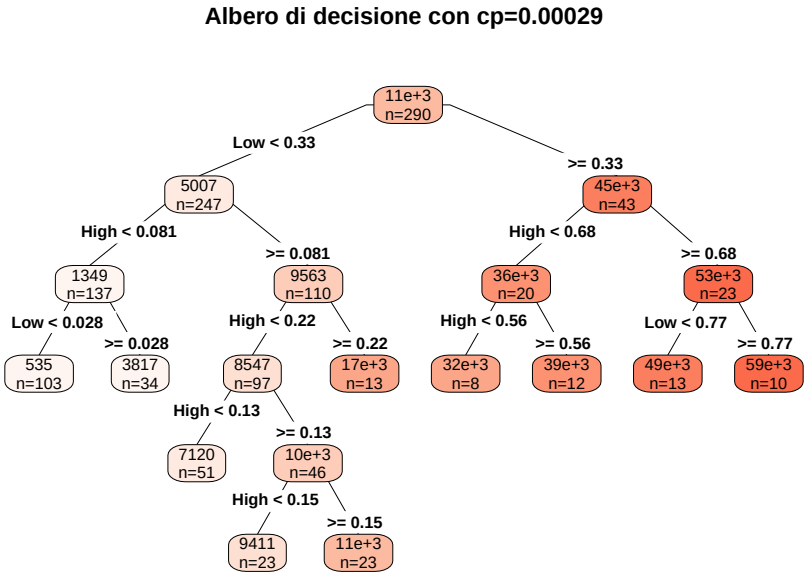

Figure 8: CART DT: tuning automatico su 10 valori

Il valore del parametro cp che ha portato alla configurazione dell'albero sopra è **cp= 0.00029** e, aumentandolo ulteriormente, non sono stati riscontrati miglioramenti significativi, come mostra la figura 9.

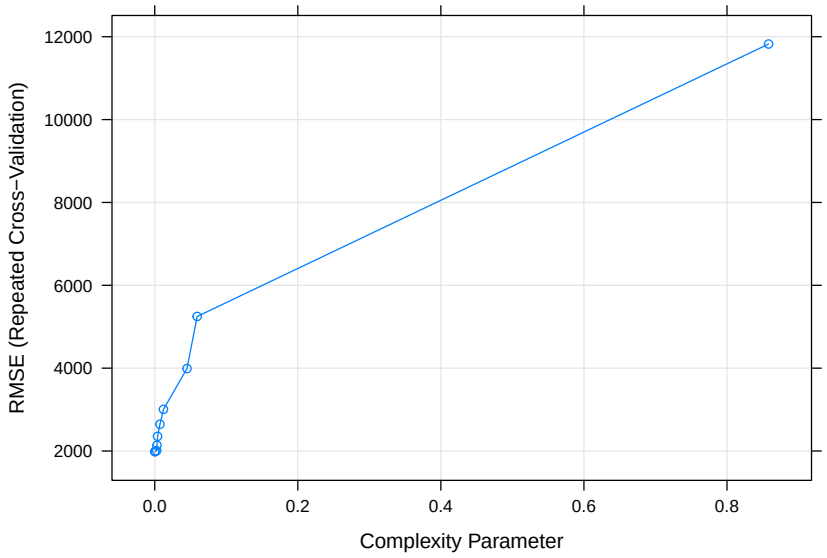

Figure 9: CART DTR: fase di tuning di cp su 10 valori

Le misure di performance per quest'ultima ottimizzazione del modello sono:

- **RMSE: 1760.3**
- **R-Squared: 0.99040**
- **MAE: 977.61**

Dunque la versione di DTR con cp=0.00029 costituisce sicuramente una miglioria rispetto a quella con cp=0.045, anche se non si è ancora riusciti ad avere performance migliori del modello MLR.

A seguire si può osservare il grafico della dispersione delle predizioni effettuate dal modello appena discusso, che risulta essere abbastanza sparso, specialmente per valori grandi.

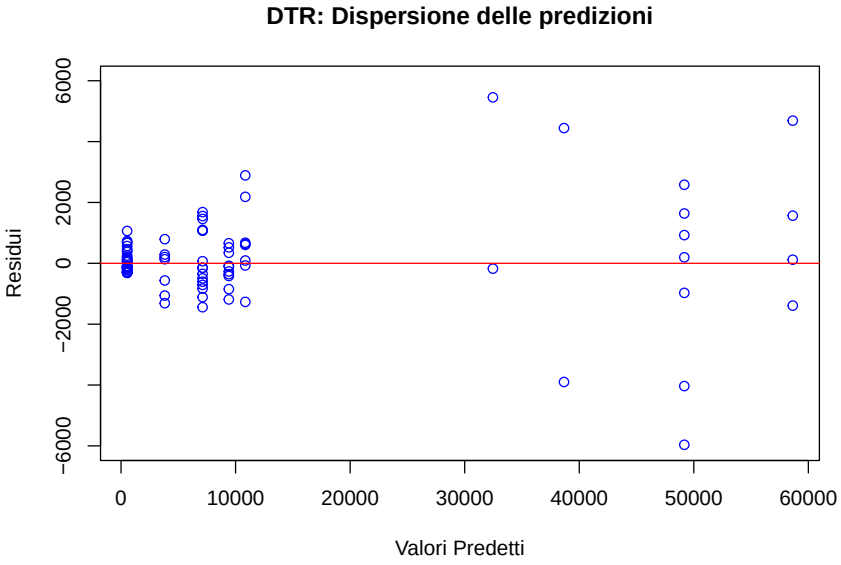

Figure 10: CART DTR con cp=0.00029: dispersione delle predizioni

3.3 Random Forest Regressor

Il Random Forest (**RF**) è un modello di classificazione o regressione ensemble che combina i risultati prodotti da diversi DT mediante la tecnica **bootstrap**. Se i DT utilizzati sono, in particolare, DTR allora si parlerà di Random Forest Regressor (**RFR**).

La tecnica bootstrap consiste nel dividere il dataset di training in **k** sottoinsiemi $\mathbf{T}_1 \ldots \mathbf{T}_k$ mediante un'operazione di sampling con possibile ripetizione, addestrare ciascun modello DT_i sul corrispondente sottoinsieme T_i ed infine combinare i risultati ottenuti. In caso di classificazione si prenderà l'etichetta di maggioranza predetta dai DT, mentre nel caso della regressione si prenderà la media dei valori predetti dai DT.

Nell'operazione di sampling le tuple contenute in un sottoinsieme T_i saranno prese casualmente da tutto il training set e potrà capitare che due sottoinsiemi T_i, T_j con $i \neq j$ avranno intersezione non vuota.

Un'altra operazione svolta dal RF nella fase di creazione del modello è quella del **bagging** sugli attributi del dataset. Similmente al bootstrapping sulle tuple, il bagging sugli attributi prevede, per ciscun DT, un training su un sottoinsieme di quest'ultimi. Questa tecnica permette di ridurre la correlazione tra i vari alberi nel caso siano presenti attributi predittori molto forti per una classe.

3.3.1 Applicazione di RF sul dataset di bitcoin

Come fatto per i precedenti modelli, si applica ora il RF al dataset del bitcoin. La stringa identificativa del modello RF da passare al metodo train() è **rf**.

Per questo modello di regressione caret ha individuato il parametro **mtry=3**, che indica il numero di attributi scelti randomicamente nel training dei vari DT, ovvero il numero di attributi su cui fare bagging.

Di seguito è raffigurato il grafico che mostra la fase di tuning, che prevede il test del parametro mtry sui valori 2, 3 e 4.

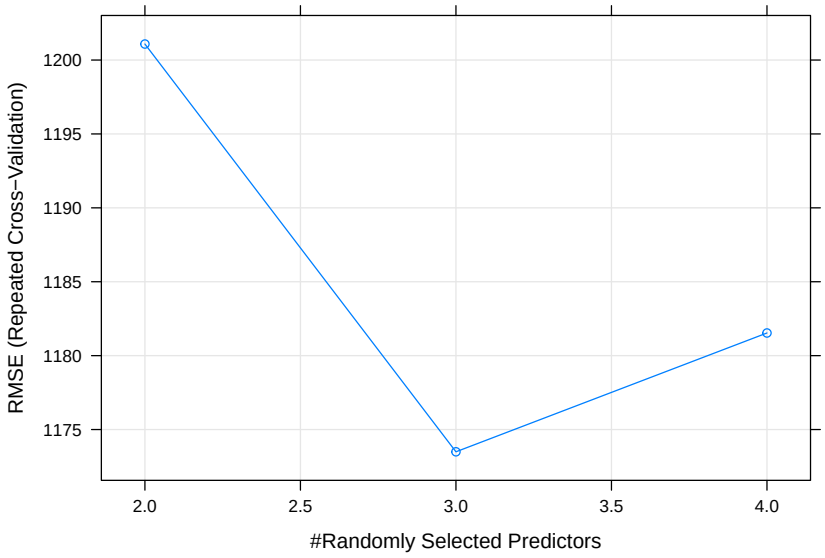

Figure 11: RF: fase di tuning

Come è immediato notare dalla figura 11 mtry=3 permette di avere il minor errore nel dataset di training, pertanto il modello sarà creato con questa impostazione.

Per quanto riguarda il parametro **n.tree**, ovvero quello che indica il numero di DT da usare, caret ha un'impostazione automatica a **500**. Si è preferito mantenere questo valore poiché dall'andamento della funzione del grafico di figura 12 si può chiaramente osservare che da un numero di alberi pari a 500 in poi non ci sono significative diminuzioni dell'errore di predizione.

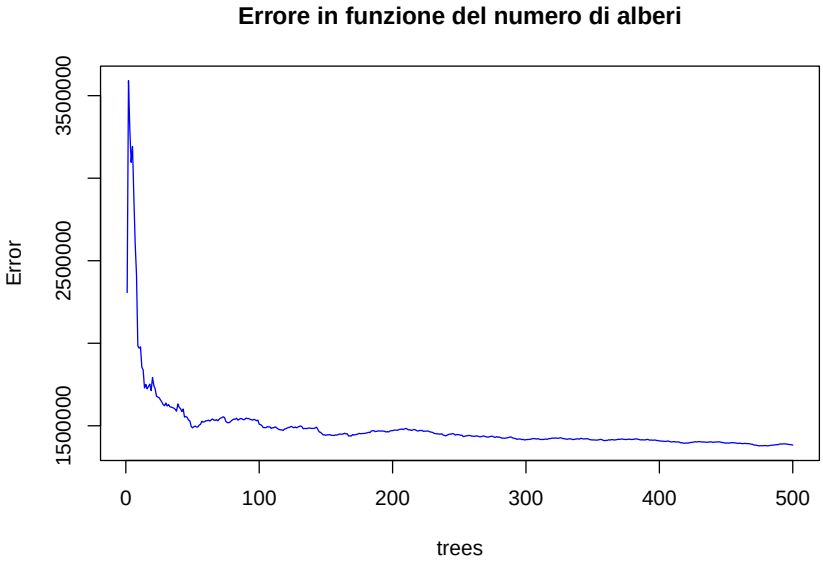

Figure 12: RF: errore di predizione in funzione del numero di DT

Per concludere vengono riportati i risultati ottenuti dalle predizioni effettuate sul test set:

- **RMSE: 1275.5**
- **R-Squared-tuned: 0.99492**
- **MAE-tuned: 563.53**

Il RFR ha prodotto risultati migliori rispetto al DTR (come ci si poteva aspettare dato che utilizza più DTR combinati), ma rimangono peggiori se confrontati con quelli del MLR.

A seguire il grafico della dispersione delle predizioni, che risulta essere, in accordo con le misurazioni ottenute, meno sparso rispetto a quello prodotto dal modello DTR, ma ancora molto distante da quello del MLR.

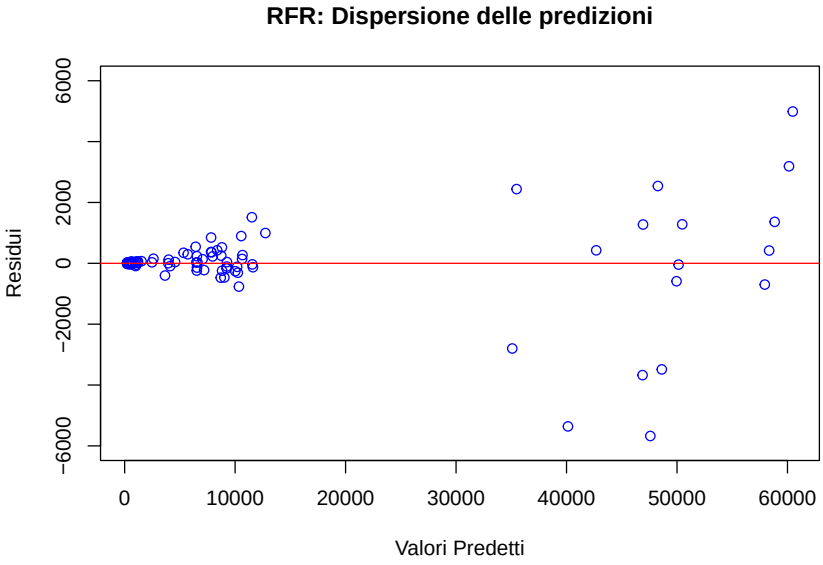

Figure 13: RF con mtry=3, n.tree=500: dispersione delle predizioni

3.4 Support Vector Regressor

Il Support Vector Regressor (SVR) è un algoritmo di regressione che permette di trovare, come nella regressione lineare, la retta (o **hyperplane**) che meglio approssima i data points, ovvero le tuple del dataset utilizzate come esempi per il modello.

Si supponga che l'hyperplane abbia equazione **wx**-b=0, con **w** vettore di coefficienti del vettore **x** e b intercetta.

Si supponga inoltre di avere a disposizione due rette **H1** e **H2** parallele e distanziate ϵ dall'hyperplane, che fungono da **decision boundaries** e creano un margine la cui larghezza è $\frac{2}{\|\mathbf{w}\|}$.

Allora l'obiettivo di SVR è quello di minimizzare la seguente quantità:

$$\frac{1}{2}\|\mathbf{w}\|^2$$

Mantenendo soddisfatto il vincolo:

$$|\mathbf{y}_i - \mathbf{w}\mathbf{x}_i| \leq \epsilon \quad \forall i$$

Dove $\mathbf{y}_i$ e $\mathbf{w}\mathbf{x}_i$ sono rispettivamente valore reale e predetto della tupla i-esima del dataset.

In questo modo si cerca l'hyperplane con il margine di ampiezza maggiore, considerando solo i data points di training che rientrano in quest'ultimo.

Il fitting può comunque essere migliorato aggiungendo **N** variabili **slack** ξ_i che tengano traccia della distanza tra i punti esterni al margine e il margine stesso.

Allora la quantità da minimizzare diventa:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}|\xi_i| \quad \text{con C} \in \mathbb{R}$$

Dove la costante C può essere usata per massimizzare il numero di data points all'interno del margine. Il vincolo diventa invece:

$$|\mathbf{y}_i - \mathbf{w}\mathbf{x}_i| \leq \epsilon + |\xi_i| \quad \forall i$$

L'hyperplane così trovato può essere utilizzato per predire valori dell'attributo target in funzione delle features, poiché costituisce proprio una retta di regressione in cui poter trovare le stime.

Una nota importante sul modello SVR è che anche nel caso in cui i data points non siano linearmente separabili li si può rendere tali con l'uso di una funzione kernel, che mappa i punti in uno spazio a più elevata dimensione, dove dalla teoria si sa che tendono a distanziarsi l'un l'altro.

3.4.1 Applicazione di SVR al dataset di bitcoin

Applicando quanto visto al caso del bitcoin, si è notato sperimentalmente che, tra le funzioni kernel **linear**, **polynomial** e **radial**, quella che ha performato meglio è stata la linear.

La creazione del modello avviene, al solito, tramite il metodo train() a cui va passata la stringa **svmLinear**, in modo da scegliere la versione di SVR con kernel linear. In alternativa si possono scegliere le versioni polynomial e radial con le stringhe **svmPoly** e **svmRadial** rispettivamente.

Nella versione linear di SVR i parametri migliori individuati da caret nella fase di tuning sono stati ϵ**=0.1** e **C=1**. Tuttavia mediante un tuning esplicito dei parametri mediante la funzione **expand.grid()**, che prende in input una lista di valori da testare per ciascun parametro del modello, si è riusciti ad individuare l'impostazione **C=0.75** che ha permesso di abbassare ulteriormente l'errore nel dataset di training. L'oggetto ritornato dal metodo expand.grid() va passato al parametro tuneGrid del metodo train(), affinché possa essere utilizzato.

Anche nella versione polynomial di SVR è stato effettuato un tuning esplicito dei parametri, in particolare l'impostazione **C=0.25**, **scale=0.5**, **degree=2** si è rivelata la migliore per il dataset a disposizione.

Nella versione radial di SVR, come per le precedenti, sono stati trovati i valori ottimali dei parametri mediante tuning espliciti e, in dettaglio, l'impostazione che ha dato migliori risultati è stata **C=5** e **sigma=0.2**.

La spiegazione dei vari parametri sopra menzionati esula dallo scopo di questa trattazione, pertanto si invitano i lettori interessati a consultare la documentazione di caret posta nella bibliografia.

I risultati delle misure di performance sulle tre versioni di SVR sono i seguenti:

- **RMSE-linear: 654.18**
- **RMSE-polynomial: 757.95**
- **RMSE-radial: 1439.4**
- **R-Squared-linear: 0.99868**
- **R-Squared-polynomial: 0.99883**
- **R-Squared-radial: 0.99587**
- **MAE-linear: 324.08**
- **MAE-polynomial: 513.91**
- **MAE-radial: 1141.4**

Come già anticipato, la versione SVR linear si è rivelata migliore delle altre due se confrontiamo i risultati di RMSE e MAE, e questo è in accordo col fatto che, come visto nella figura 4 della sezione dedicata al modello MLR,

molte features sono correlate linearmente con l'attributo target Close, dunque un modello lineare come SVR linear è adatto a cogliere questo tipo di relazioni, portando a predizioni più accurate.

In figura 14 si può osservare l'andamento dell'errore in funzione del parametro C nel caso di SVR linear.

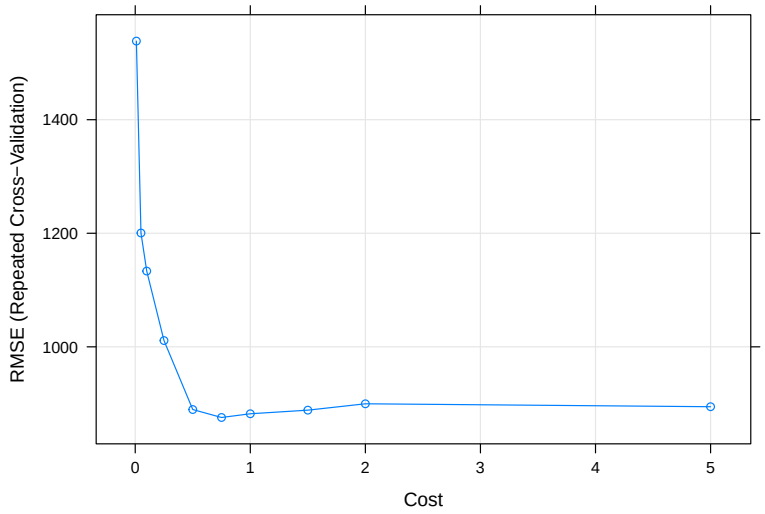

Figure 14: SVM linear: fase di tuning dei parametri

In figura 15 e 16 si possono osservare invece l'andamento dell'errore in funzione dei parametri C, scale, degree e sigma dei modelli SVR polynomial e SVR radial.

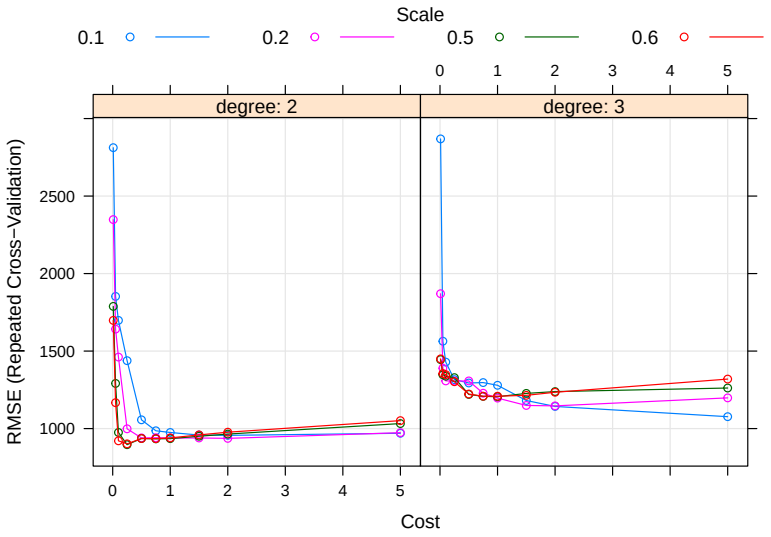

Figure 15: SVM polynomial: fase di tuning dei parametri

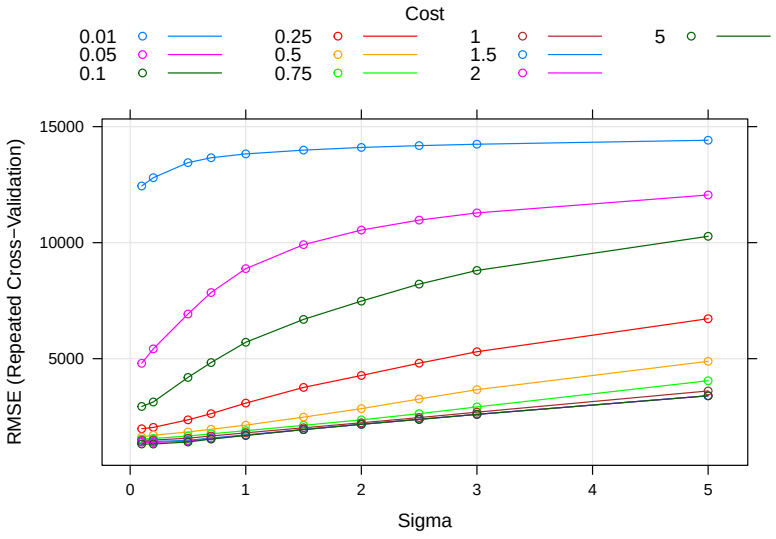

Figure 16: SVM radial: fase di tuning dei parametri

A titolo conclusivo dello studio del modello SVR si riportano i grafici di dispersione delle predizioni per le tre versioni.

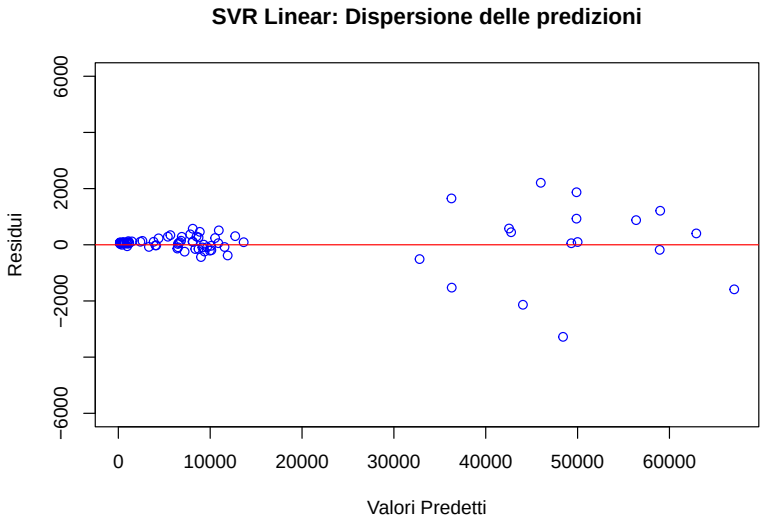

Figure 17: SVM linear: dispersione delle predizioni

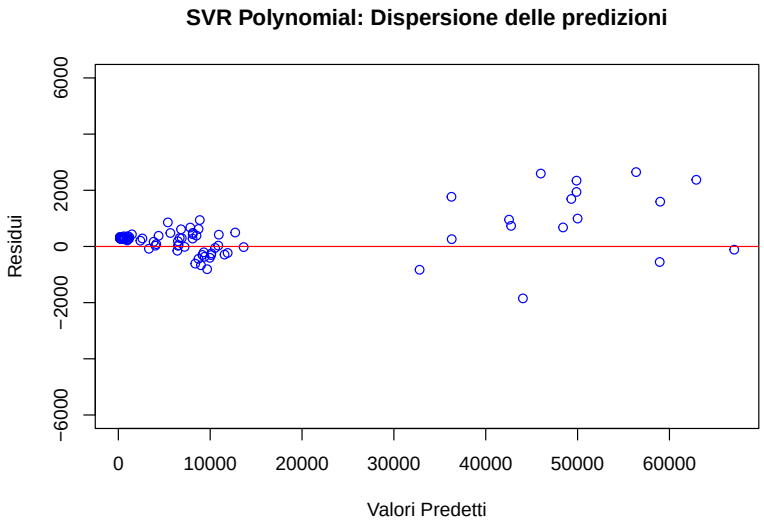

Figure 18: SVM polynomial: dispersione delle predizioni

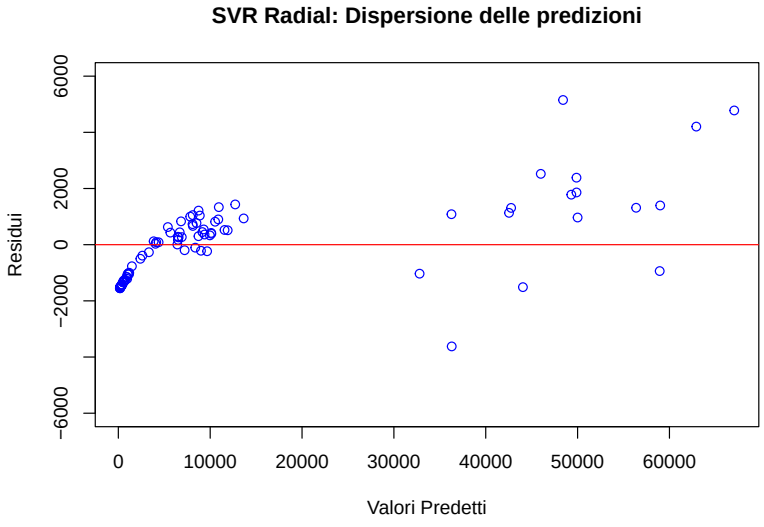

Figure 19: SVM radial: dispersione delle predizioni

Come da misure di performance trovate, si può osservare come la dispersione più leggera è locata nel grafico 17 del modello SVR linear, che presenta errore di predizione minore, mentre la più marcata la si individua nella figura 19 del modello SVR radial, che invece presenta un maggior errore di predizione.

4 Conclusioni

L'analisi e l'applicazione dei quattro modelli di regressione trattati, sul dataset del bitcoin, ha portato a dei risultati molto interessanti, che hanno permesso di individuare non solo l'algoritmo che ha performato meglio sul dataset di test, ma anche la categoria di regressori che possono ritenersi validi per la risoluzione di questo preciso problema. Si sta parlando proprio dei modelli lineari che, come già detto in precedenza, hanno un notevole vantaggio in quanto la correlazione tra le coppie (High,Close), (Low,Close) e (Open,Close) risulta essere lineare.

Non resta che proclamare il modello di regressione migliore, ovvero il Multiple Linear Regression con RMSE=567.24, R-Squared=0.9990 e MAE=281.22.

Nota di merito anche per il modello SVR con kernel linear che ha ottenuto RMSE= 627.89, R-Squared=0.99878 e MAE=310.96, piazzandosi come secondo miglior regressore per il dataset in questione.

Per concludere questa trattazione, si vuole far notare come i risultati ottenuti difficilmente possano trovare una reale applicazione. Questo perché tutta l'analisi affrontata si è basata su uno storico di dati proveniente dal bitcoin, quindi da una sua storia passata che, non è detto si ripeta in futuro. Inoltre esistono molti altri fattori che influenzano il prezzo del bitcoin, come quelli di natura fondamentale, si pensi a decisioni politiche come quella dello stato di El Salvador nell'introduzione del bitcoin come valuta officiale del paese, decisioni di business prese da grosse aziende come MasterCard e Visa o ancora tweet di persone influenti come Elon Musk e così via.

Nonostante questa consapevolezza, tutte le tecniche qui adottate rimangono valide e possono sempre essere prese come spunto per l'analisi di dati di più realistico utilizzo.

References

[1] https://it.wikipedia.org/wiki/Bitcoin

[2] https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

[3] https://www.educba.com/support-vector-regression/

[4] https://topepo.github.io/caret/index.html

[5] https://www.kaggle.com/varpit94/bitcoin-data-updated-till-26jun2021

[6] https://www.analyticsvidhya.com/blog/2021/05/5-regression-algorithms-you-should-know-introductory-guide/

[7] slides del corso "Introduzione al Data Mining" di Giovanni Micale