



UNIVERSITÀ DEGLI STUDI DI PERUGIA  
Dipartimento di Matematica e Informatica



TESI DI LAUREA TRIENNALE IN INFORMATICA

# Bitcoin price-trend prediction using Machine Learning

*Candidato*

**Cristian Cosci**

*Relatore*

**Prof. Francesco Santini**

---

Anno Accademico 2020-2021

# Indice

<b>1</b>	<b>Introduzione</b>	<b>6</b>
<b>2</b>	<b>Le Criptovalute e il Bitcoin</b>	<b>8</b>
2.1	Introduzione alle Criptovalute . . . . .	8
2.2	Il Bitcoin . . . . .	10
2.2.1	Struttura di un Blocco . . . . .	12
2.2.2	Transazioni . . . . .	15
2.2.3	Mining . . . . .	16
<b>3</b>	<b>Analisi Tecnica e Analisi Fondamentale</b>	<b>18</b>
3.1	I principali indicatori tecnici finanziari . . . . .	18
3.2	L'analisi fondamentale nel caso del Bitcoin . . . . .	20
<b>4</b>	<b>Il Machine Learning</b>	<b>23</b>
4.1	Introduzione al Machine Learning . . . . .	23
4.1.1	I tipi di apprendimento . . . . .	24
4.1.2	Classificazione e Regressione . . . . .	25
4.2	Random Forest . . . . .	27
4.2.1	Alberi decisionali . . . . .	27
4.2.2	Bagging . . . . .	31
4.2.3	Dal Bagging alle Random Forests . . . . .	31
4.3	Support Vector Machine . . . . .	34
4.3.1	Classificazione . . . . .	34
4.3.2	Regressione . . . . .	39
4.4	Deep Learning e Artificial Neural Network . . . . .	41

4.4.1	Perceptron . . . . .	41
4.4.2	Multilayer Artificial Neural Network . . . . .	45
4.4.3	Long Short-Term Memory . . . . .	49
4.5	Overfitting . . . . .	54
4.6	Valutazione di un modello . . . . .	57
<b>5</b>	<b>Ricerche correlate</b>	<b>60</b>
5.1	La teoria del mercato efficiente . . . . .	60
5.2	Ricerche correlate . . . . .	62
<b>6</b>	<b>Ricerca ed Analisi</b>	<b>65</b>
6.1	Raccolta dati . . . . .	65
6.1.1	Descrizione Feature . . . . .	66
6.2	Realizzazione Dataset di allenamento . . . . .	70
6.2.1	Feature Selection . . . . .	71
6.2.2	Boruta . . . . .	73
6.2.3	Correlazione di Pearson e Spearman . . . . .	75
6.2.4	Feature selezionate . . . . .	78
6.3	Pre-processing dei dati . . . . .	87
6.3.1	Normalizzazione e Standardizzazione . . . . .	88
6.3.2	Dati di training e test . . . . .	91
6.4	Sistema di simulazione di trading . . . . .	92
6.5	Random Forest . . . . .	95
6.5.1	Classificatore . . . . .	96
6.5.2	Regressore . . . . .	99
6.5.3	Analisi dei modelli su time frame ampio . . . . .	99
6.6	Support Vector Machine . . . . .	113
6.6.1	Classificatore . . . . .	114
6.6.2	Regressore . . . . .	116
6.6.3	Analisi dei modelli su time frame ampio . . . . .	117
6.7	Long Short-Term Memory . . . . .	125
6.7.1	Implementazione e test . . . . .	125
6.7.2	Sistema di voting multi-modello . . . . .	129

<b>7 Conclusioni</b>	<b>132</b>
7.1 Analisi dei risultati . . . . .	132
7.2 Lavori futuri . . . . .	134

# Capitolo 1

## Introduzione

L'avvento delle criptovalute è alquanto recente, basti pensare che sono passati solo poco più di 11 anni dalla nascita di Bitcoin. Dal 2009 in poi numerosi altri progetti sono nati, e l'affluenza di persone in questo mercato è cresciuta esponenzialmente. Recentemente, l'intero mercato delle criptovalute ha raggiunto una capitalizzazione di mercato di oltre 2 triliardi di dollari [14]. Bitcoin, dalla sua nascita, è passato a valere da pochi centesimi di dollaro a oltre 60000\$. Molte persone hanno intravisto il vero potenziale delle criptovalute, mentre altre hanno preso in considerazione questo mondo solo come possibile fonte di guadagno. Qualunque sia il fine, che sia speculazione o un vero e proprio investimento, il mercato delle criptovalute è cresciuto a dismisura. Nell'ultimo periodo, moltissime aziende e grandi investitori hanno iniziato ad allocare parte del loro portafoglio in valute digitali. Ciò ha suscitato ancor più interesse tra la popolazione. Ad oggi il mercato delle criptovalute è altamente volatile e possono avvenire grandi variazioni di prezzo in poche ore, a volte anche minuti. Bitcoin è passato dall'essere preso in considerazione come mezzo di pagamento, all'essere acquistato a fini d'investimento e riserva di valore, proprio come l'oro e altri minerali preziosi. Bitcoin è infatti da molti ritenuto una forma di oro digitale, ovvero un asset in grado di proteggere l'investimento dall'inflazione. Dato l'interesse crescente, numerose strategie di previsione del prezzo delle criptovalute, o del suo trend, sono poste sotto la lente degli studiosi e ricercatori. A partire da semplici approcci, si è passati all'applicazione di tecniche di Machine Learning.

Il seguente lavoro di tesi, è volto allo studio del possibile utilizzo di tecniche di apprendimento automatico, nell'ambito della previsione dell'andamento di prezzo del

Bitcoin. L’obiettivo è di fornire una base a successivi studi in materia, ulteriori implementazioni e migliorie. L’idea generale è quella di implementare modelli predittivi per un tipico sistema in grado di gestire automaticamente un’allocazione di portafoglio in Bitcoin, prevedendo l’andamento dei mercati e effettuare in autonomia le continue azioni di compravendita. In particolare, oltre a richiedere al sistema di generare profitto, il focus principale è quello di mantenere e salvaguardare il capitale, proteggendolo da ingenti perdite e grosse fluttuazioni negative. Sono quindi di primaria importanza le prestazioni del modello. Numerosi modelli di apprendimento automatico sono stati implementati e testati. Per allenare i modelli sono stati utilizzati indicatori tecnici finanziari e numerosi dati on-chain (provenienti direttamente dalla blockchain di Bitcoin). Nel Capitolo 2 è presentata la tecnologia alla base delle criptovalute ed in particolare di Bitcoin. Nel Capitolo 3 vi è una breve panoramica sull’analisi tecnica e sull’analisi fondamentale. Nel Capitolo 4 sono presentati e descritti gli argomenti di principale importanza di questa tesi, tra cui un’introduzione al Machine Learning e ai modelli utilizzati. Nel Capitolo 5 è riportata una breve descrizione dello stato attuale delle ricerche in tale ambito. Nel Capitolo 6 verranno esposte tutte le tecniche e gli approcci utilizzati, nonché i modelli implementati e i risultati ottenuti. Infine, nel Capitolo 7 saranno analizzati i risultati e tratte le conclusioni della ricerca.

# Capitolo 2

## Le Criptovalute e il Bitcoin

Con il termine **Criptovaluta** (in inglese *cryptocurrency*), ci si riferisce ad una rappresentazione digitale di valore inteso come moneta. Si tratta di un sistema basato sulla crittografia, volto a fornire un mezzo di scambio. Si può parlare quindi di moneta o valuta digitale, la cui implementazione permette di effettuare pagamenti per beni e servizi. A partire dal 2009 l'utilizzo delle criptovalute è andato progressivamente crescendo. Quest'ultime, non venendo riconosciute da nessuna autorità nazionale, hanno un valore dato solamente da quello che gli utilizzatori sono disposti ad attribuirgli mediante la regola "*domanda-offerta*". In questo modo, il valore assunto, risulta essere quello che le persone sono disposte ad *accettare per liberarsene o a pagare per attribuirsiene*.

### 2.1 Introduzione alle Criptovalute

La tecnologia che ne sta alla base è chiamata **Blockchain** (catena di blocchi), la quale costituisce una struttura dati condivisa ed immutabile, la cui integrità è garantita dall'uso della crittografia (da qui l'utilizzo della parola *crypto* in criptovaluta). La si può immaginare come un grande registro digitale distribuito, nel quale le informazioni sotto forma di blocchi, sono concatenate in ordine cronologico. Ogni blocco contiene un puntatore hash come collegamento al blocco precedente, una marcatura temporale e dati sulle transazioni (vedi Sezione 2.2.1). Le informazioni non sono detenute in un unico computer (*nodo*), ma su insieme di essi, si parla quindi di rete **peer-to-peer**. La consultazione di tale registro è libera a tutti, mentre la scrittura/modifica (ag-

giunta di un nuovo blocco) è *regolata da un protocollo condiviso dalla rete e dai nodi che ne fanno utilizzo*.

L'aggiunta di un nuovo blocco, una volta autorizzato, porta alla modifica e all'aggiornamento della catena su ogni nodo, rendendolo così pubblico ed immutabile. Essendo un registro distribuito e di aperta consultazione, ogni utente ha la possibilità di controllare, tracciare e verificare le informazioni; non si necessita di intermediari o enti centrali che debbano garantire veridicità dei dati, garantire per conto di altri o dettare regole. Molto spesso la tecnologia Blockchain viene confusa con il concetto di criptovalute, in realtà non si tratta della medesima cosa. La Blockchain, e la sua ideazione, risale a molto prima dell'avvento delle criptovalute, anche se il primo utilizzo e la sua prima implementazione è stata attuata con quest'ultime. Le criptovalute sono a tutti gli effetti delle monete decentralizzate, nate per essere slegate da qualsiasi supervisione e influenza governativa di banche centrali o entità pubbliche, caratteristiche che la Blockchain permette a pieno. Volendo dare una definizione in grado di incarnare i concetti essenziali, una criptovaluta è un sistema che soddisfa sei condizioni [45]:

1. Il sistema non richiede un'autorità centrale, il suo stato è mantenuto attraverso un consenso distribuito.
2. Il sistema mantiene un controllo delle unità di criptovaluta e della loro proprietà.
3. Il sistema determina se possono essere create nuove unità di criptovaluta. Se tali unità si possono creare, il sistema definisce la loro origine e come determinare il loro possessore.
4. La proprietà di una criptovaluta può essere provata solo crittograficamente.
5. Il sistema consente di eseguire transazioni nelle quali avviene un cambio di proprietà delle unità crittografiche. La conferma della transazione può essere rilasciata solo da un ente che può provare la proprietà delle criptovalute oggetto della transazione.
6. Se vengono date simultaneamente due diverse istruzioni per il cambio di proprietà delle stesse unità crittografiche, il sistema esegue al massimo una delle due.

Al giorno d'oggi esistono oltre 1000 criptovalute con funzionalità, scopi, protocolli e sistemi di validazione e verifica dei blocchi differenti. Per la convalida dei blocchi, si hanno progetti che adoperano algoritmi di **Proof of Work** (PoW), nel quale dei super calcolatori, cercano di risolvere un problema matematico complesso, al fine di poterne trovare la soluzione (una stringa hash) e ricevere una ricompensa (in criptovaluta) per il lavoro svolto. Questo processo è svolto dai cosiddetti "**miners**" (minatori), i quali operano in competizione per raggiungere la soluzione al problema per primi (vedi Sezione 2.2.3). Una volta che ciò si verifica, il blocco viene convalidato e aggiunto alla catena, previo consenso da parte dei nodi, i quali devono verificare che l'hash trovato come soluzione sia effettivamente corretto (è necessario che sia verificato dal 50+1% della rete mediante il sistema "*Una CPU un voto*" [53]). Altre criptovalute adoperano invece algoritmi computazionalmente meno costosi, chiamati **Proof of Stake** (PoS), il cui obiettivo è il medesimo, convalidare i blocchi e garantire che siano effettivamente corretti. In questo caso non si ha una competizione tra i minatori, ma esistono i cosiddetti "**validatori**", grandi detentori di criptovalute, che garantiscono la validità delle operazioni effettuate impegnando una quota di esse ("*stake*"). Altre sostanziali differenze tra i vari progetti sono rivolte all'utilizzo che se ne fa, alcune criptovalute possono essere utilizzate come solo metodo di pagamento e/o scambio digitale, mentre in altri casi abbiamo dei veri e propri ecosistemi decentralizzati le cui applicazioni principali sono:

- Esecuzione di "Contratti Intelligenti" (**Smart Contracts**).
- Sistemi di finanza decentralizzata (**DeFi**).
- Token non fungibili per arte ed articoli da collezione (**NFT**).
- Videogiochi (**DeFi Gaming**).

## 2.2 Il Bitcoin

Bitcoin (abbreviazione BTC) è la prima criptovaluta sviluppata, nata nel gennaio 2009. L'idea viene presentata l'anno prima da Satoshi Nakamoto (*pseudonimo dello sviluppatore o team di sviluppatori*) nel White paper "*Bitcoin: A Peer-to-Peer Electronic Cash System*" [53]. Ad oggi è la criptovaluta più conosciuta, prima per capita-

lizzazione di mercato e per valore, nonostante dal giorno della sua nascita numerose altre monete digitali siano state coniate e sviluppate. Nasce come risposta alla crisi finanziaria del 2007-2008, con l'obiettivo di condurre transazioni senza l'intervento delle istituzioni finanziarie e del governo, distaccandosi dalle valute fiat, proponendo un sistema per le transazioni elettroniche non basato sulla fiducia [53]. Bitcoin ha un importo di monete coniabili che tende asintoticamente a 21 milioni di unità [14] (vedi Figura 2.1).

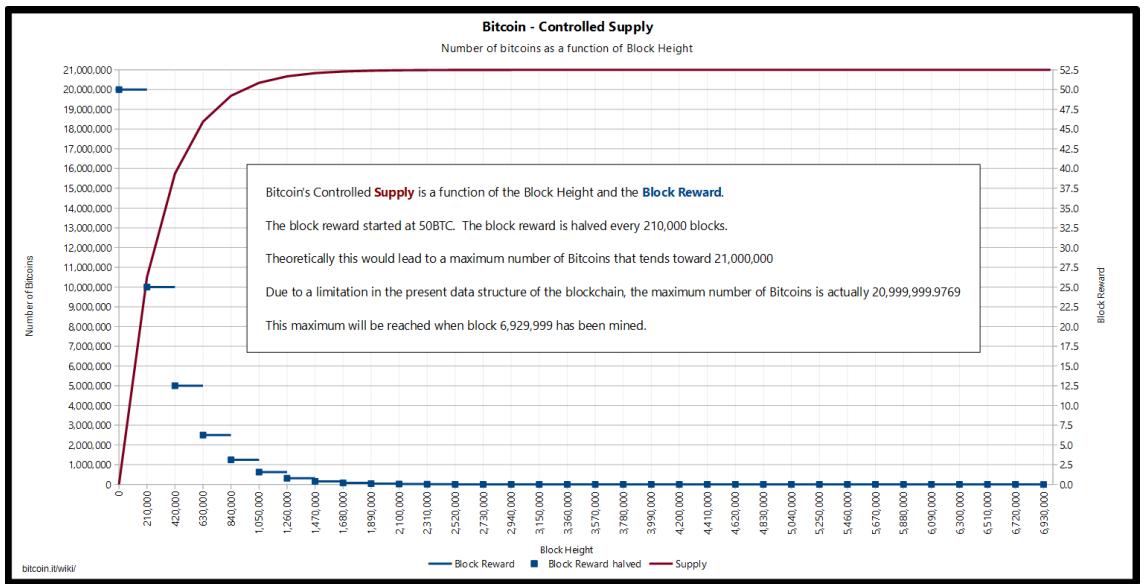


Figura 2.1: Max supply di Bitcoin [8].

La rete utilizza la Proof of Work come sistema di convalida dei blocchi e i minatori ricevono ricompense in BTC ad ogni blocco minato. Tali ricompense vengono dimezzate ogni 4 anni, fino ad arrivare al momento in cui l'unico incentivo al lavoro per i miners sarà derivante dalle commissioni, pagate dagli utenti per effettuare transazioni. Inoltre la complessità dei problemi da risolvere, del sistema PoW, si adatta automaticamente alla potenza di calcolo fornita dall'intera rete, in modo da rendere il sistema sostenibile e duraturo nel tempo (vedi Sezione 2.2.3).

### 2.2.1 Struttura di un Blocco

Gli elementi fondamentali della Blockchain sono i **blocchi**, infatti quest'ultima può essere visualizzata come uno stack verticale, con blocchi sovrapposti l'uno sull'altro, con il primo di essi che funge da base. La distanza dal primo blocco ad un altro, nella pila, è detta *altezza*. Ogni blocco all'interno della catena è identificato da un hash, che nel caso del sistema Bitcoin, è generato utilizzando l'algoritmo crittografico SHA256. Ogni blocco fa riferimento al blocco precedente, noto come blocco padre, tramite un apposito campo nell'intestazione del blocco stesso. La catena di blocchi è identificata dalla sequenza di hash che collega ciascun blocco al padre, fino al primo blocco mai creato, noto come *blocco di genesi* [4]. Nel caso in cui un blocco padre venga modificato, il suo hash viene alterato. Ciò richiede un cambiamento nel puntatore al blocco precedente del blocco figlio, questo fa sì che, a sua volta, tutta la catena debba essere modificata. L'effetto a cascata che si genera, assicura che nel momento in cui un blocco viene susseguito da numerosi altri, una sua modifica diventi impossibile senza forzare un ricalcolo di tutti i blocchi successivi. Poichè un tale ricalcolo richiederebbe un costo enorme, l'esistenza di una lunga catena rende immutabile la blockchain, una caratteristica chiave della sicurezza di Bitcoin.

Un blocco è una struttura dati che consente di aggregare le transazioni. Ciascun blocco è costituito da un'**intestazione**, contenente i metadati, seguita da un lungo elenco di transazioni. L'intestazione è di 80 byte, mentre una transazione media è di almeno 250 byte, consentendo al blocco di contenere più di 500 transazioni [4].

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header (see below)
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Figura 2.2: Struttura di un blocco della Blockchain Bitcoin [4].

L'intestazione del blocco è costituita da tre set di metadati:

- un riferimento all'hash del blocco precedente;
- difficoltà, timestamp e nonce (utilizzati per il mining);

- radice del Merkle Tree (o Binary Hash Tree), una struttura dati basata su alberi binari contenenti hash crittografici, utilizzata per raggruppare in modo efficiente le transazioni.

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the Merkle-Tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

Figura 2.3: Struttura dell'*header* di un blocco della Blockchain Bitcoin [4].

L'identificatore principale di un blocco è il suo **hash crittografico**, un'impronta digitale, creata eseguendo l'hashing dell'intestazione due volte, tramite l'algoritmo SHA256. L'hash risultante a 32 byte è chiamato *hash del blocco*, ma è più precisamente l'hash dell'intestazione, poiché solo quest'ultima viene utilizzata per calcolarlo. L'hash identifica un blocco in modo univoco e non è effettivamente incluso all'interno della struttura dati, né quando il blocco viene trasmesso sulla rete, né quando viene archiviato nella memoria di persistenza di un nodo come parte della blockchain. Un secondo modo per identificare un blocco è dalla sua posizione nella blockchain, ovvero dalla sua altezza [4].

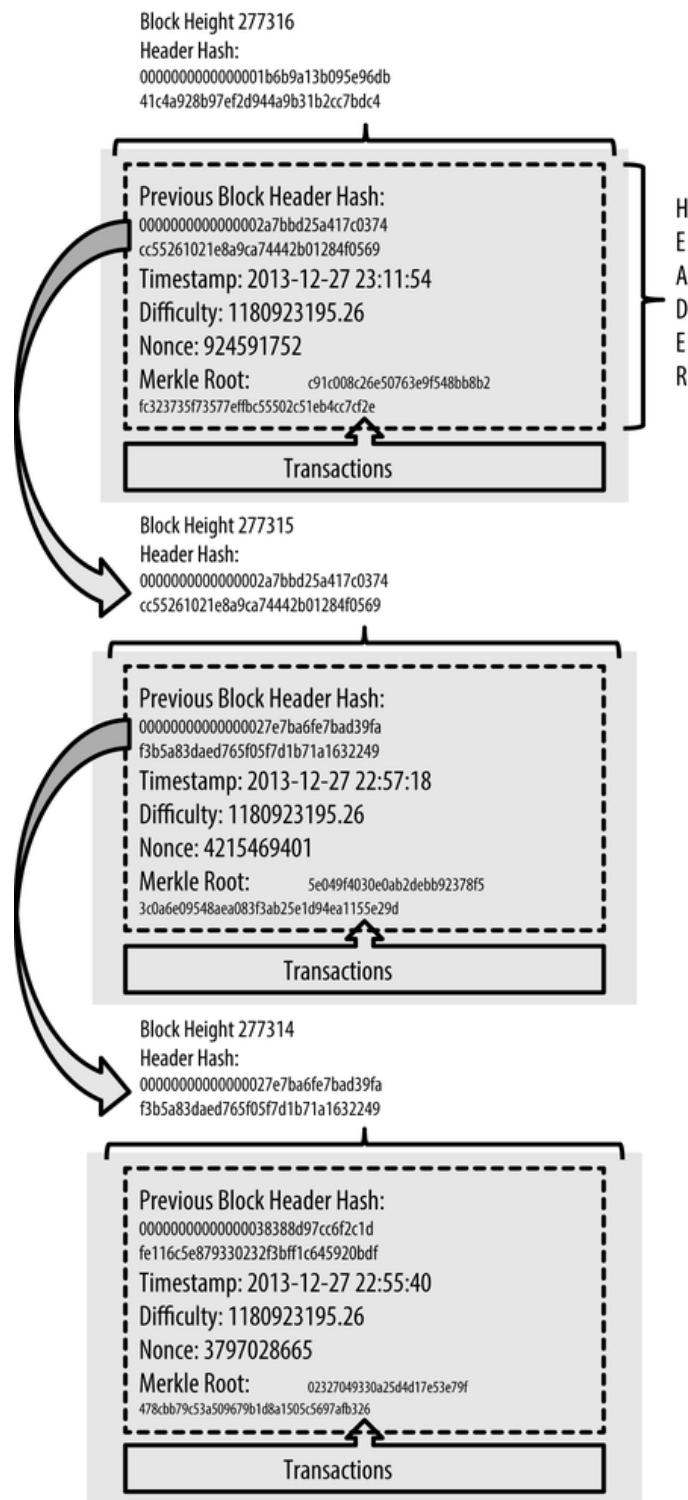


Figura 2.4: Esempio di blockchain [4].

## 2.2.2 Transazioni

Nella tecnologia Bitcoin, "*spendere*" significa firmare una transazione che trasferisce valore da una transazione precedente ad un nuovo proprietario. Solo nel momento in cui una transazione viene inserita nella blockchain è considerata valida. Le transazioni vengono aggiunte al nuovo blocco, creato in media ogni dieci minuti, in base ad un sistema di priorità (ad esempio quelle con commissione più alta vengono aggiunte per prime). Bitcoin, basandosi su un algoritmo PoW, richiede un'enorme quantità di calcolo per far sì che le transazioni, raggruppate in blocchi, vengano inserite nella blockchain. Tuttavia è richiesta solo una piccola capacità computazionale per far sì che un blocco, una volta minato, venga verificato dai nodi della rete.

Una transazione è una struttura dati che consente di codificare un trasferimento di valore da una fonte, chiamata input, a una destinazione, chiamata output. Input e output non sono correlati a conti o identità, ma sono pensati come *pezzi* di moneta, bloccati con una chiave segreta, che consente al solo proprietario della chiave di farne utilizzo.

Una volta registrata sulla blockchain, la transazione è parte permanente del registro bitcoin ed è accettata come valida da tutti i partecipanti [4]. I fondi assegnati a un nuovo proprietario possono quindi essere spesi in una nuova transazione, estendendo la catena di proprietà, e ricominciando il ciclo. Poiché la transazione è firmata e non contiene informazioni riservate, chiavi private o credenziali, può essere trasmessa pubblicamente utilizzando qualsiasi trasporto di rete sottostante. A differenza delle transazioni con carta di credito, che contengono informazioni sensibili e possono essere trasmesse solo su reti crittografate.

L'elemento fondamentale è l'*output di transazione non speso o UTXO*, il quale indica la quantità di cambio di criptovaluta rimanente dopo aver eseguito una transazione. Il meccanismo è simile al resto ricevuto dopo aver effettuato una transazione in contanti presso un negozio. Ogni volta che un utente riceve bitcoin, tale importo viene registrato all'interno della blockchain come UTXO. Non esiste un saldo memorizzato di un indirizzo bitcoin o di un account, ma ci sono solo UTXO sparsi, bloccati a proprietari specifici. Il concetto di saldo bitcoin relativo ad un utente è un costrutto derivato. Il saldo viene calcolato dall'applicazione wallet, la quale scansionando la blockchain, aggrega tutti gli UTXO appartenenti a quell'utente. L'unità di misura di un UTXO, in Bitcoin, ha un valore corrispondente ad un multiplo di *satoshi*. Proprio

come i dollari possono essere divisi fino a due cifre decimali (centesimi), i bitcoin possono essere suddivisi fino a otto cifre decimali, i *satoshi*. Un UTXO è indivisibile (come una banconota) e deve essere speso nella sua interezza. Se un UTXO risulta essere più grande rispetto al valore da spendere per una transazione, vengono generati due output: uno verso il destinatario e uno come resto, verso il mittente.

### 2.2.3 Mining

Il *mining* ha due scopi principali:

- creare nuovi bitcoin ad ogni blocco;
- mantenere un sistema di fiducia che assicura la conferma delle transazioni solo se al blocco, che le contiene, è stata dedicata sufficiente potenza di calcolo.

L'algoritmo Proof-of-Work di Bitcoin consiste nell'hashing ripetuto dell'intestazione del blocco e di un numero casuale (con l'algoritmo crittografico SHA256), fino a quando non emerge una soluzione che corrisponde a un modello predeterminato. Il primo miner che trova una soluzione vince il round della competizione, e pubblica quel blocco nella blockchain. Come premio per aver trovato la soluzione, della nuova moneta viene ricevuta dal miner, mediante una speciale transazione chiamata *coinbase*. Le transazioni includono anche una commissione, che ha due scopi: incentivare i miners a processare e convalidare la transazione in un nuovo blocco; prevenire e disincentivare transazioni di spam e qualsiasi tipo di abuso del sistema. A seguito della scoperta di un nuovo blocco da parte di un miner, avviene la convalida indipendente di tale blocco, da parte di ogni nodo della rete. Il costo computazionale richiesto per la verifica è pressochè nullo in confronto a quanto richiesto per minare il blocco stesso. Nel momento in cui la maggioranza della rete ha convalidato il blocco, esso viene inserito nella blockchain, fornendo la ricompensa al miner che l'ha trovato. In questo modo solo i miners che agiscono onestamente vengono ricompensati, questo costituisce la base della Proof-of-Work, in quanto il costo necessario a manomettere la blockchain, da parte di un attaccante, risulta essere superiore al possibile guadagno ottenuto. Il sistema di generazione di nuova moneta si basa sull'estrazione dei metalli preziosi. La quantità di bitcoin creata ad ogni blocco diminuisce approssimativamente ogni quattro anni (o precisamente ogni 210.000 blocchi). Inizialmente venivano creati 50

bitcoin per blocco nel gennaio del 2009 e si è dimezzato a 25 bitcoin per blocco nel novembre del 2012, 12,5 bitcoin per blocco nel 2016, 6,25 nel 2020. Sulla base di questa formula, nell'anno 2140 tutti i bitcoin (20.9999998 milioni) saranno stati emessi. Dopo il 2140, nessun nuovo bitcoin verrà emesso e la remunerazione per i miners sarà derivante dalle sole commissioni relative alla transazioni. Secondo l'algoritmo alla base della tecnologia Bitcoin, il tempo necessario al mining di nuovi blocchi deve rimanere costante nel tempo. Quindi, la difficoltà del problema PoW si regola a seconda della potenza di calcolo della rete stessa. In questo modo anche l'aumentare della potenza di calcolo, permessa dall'avanzare della tecnologia e degli utenti che si aggiungono alla rete, rende il sistema difficilmente attaccabile e resistente a manomissione. Il ricalcolo della difficoltà avviene su ogni nodo automaticamente ogni 2016 blocchi (circa ogni due settimane): se la rete trova blocchi più velocemente di ogni 10 minuti, la difficoltà aumenta; se la rete trova blocchi più lentamente del previsto, la difficoltà diminuisce.

La tecnologia utilizzata nell'implementazione di Bitcoin è considerata ancora ad oggi tra le più sicure ed inattaccabili tra le criptovalute.

# Capitolo 3

## Analisi Tecnica e Analisi Fondamentale

Il trading è un'attività di compravendita di diversi strumenti finanziari, volta ad ottenere un profitto. Qualunque sia l’ambito (*azioni, obbligazioni, valute o materie prime*), ogni individuo, prima di operare, esegue un’**analisi** dettagliata, con l’obiettivo di individuare le azioni da eseguire. Ovviamente non è possibile determinare con certezza l’andamento dei mercati, tuttavia una buona analisi permette di ridurre gli errori. Lo studio dei mercati avviene tramite l’impiego dell’**analisi tecnica** e dell’**analisi fondamentale**. Alcuni traders si specializzano solamente in una delle due, anche se sarebbe preferibile farne un uso congiunto.

### 3.1 I principali indicatori tecnici finanziari

L’analisi tecnica si basa sullo studio dell’andamento dei prezzi, con lo scopo di prevederne le future oscillazioni. Principalmente vengono utilizzati metodi grafici e statistici, analizzando l’andamento passato si cerca di individuare eventuali pattern, trend o cicli ripetibili in futuro. La principale convinzione di questa analisi, sta nel pensare che il prezzo di un determinato asset rifletta tutte le dinamiche di mercato. Tendenzialmente l’analisi tecnica risulta essere soggettiva e, nonostante si basi su calcoli e metodi statistici, il modo in cui i risultati vengono interpretati può cambiare. Nello studio di un grafico eventuali pattern possono essere individuati da un analista piuttosto che da un altro. Allo stesso modo nel calcolo degli indicatori, potendo mo-

dificare i parametri come meglio si crede, il risultato ottenuto cambia a seconda di essi. Come per gli strumenti finanziari, anche le criptovalute dispongo di servizi di cambio (*Exchange*), disponibili sia in forma "centralizzata" come nel caso di Binance, Coinbase, Kraken e Crypto.com, sia in forma "decentralizzata" come Uniswap. In entrambi i casi gli utenti hanno a disposizione molteplici coppie di scambio, grafici, dati storici e prezzi in tempo reale. È quindi possibile eseguire un'analisi tecnica anche per le criptovalute, disponendo delle stesse informazioni che si hanno nei mercati tradizionali (prezzi di *apertura*, *chiusura*, *massimo*, *minimo* e *il volume di scambio*). Nell'implementazione del sistema di trading, per il progetto di questa tesi, come dati per l'allenamento dei modelli sono stati utilizzati alcuni tra i principali indicatori tecnici finanziari qui di seguito riportati.

**Media Mobile Semplice (SMA).** Corrisponde alla media aritmetica dei prezzi di chiusura degli ultimi N periodi (*orario*, *giornaliero*, *settimanale* etc.). Uno degli inconvenienti di questo indicatore è quello di tenere in uguale considerazione i prezzi più remoti rispetto a quelli immediatamente recenti. I principali indicatori sulla media mobile utilizzati sono calcolati su un intervallo di 50 e 200 periodi [35].

**Media Mobile Esponenziale (EMA).** Corrisponde alla media ponderata dei prezzi di chiusura degli ultimi N periodi. Il peso assegnato ad ogni elemento viene attribuito secondo un valore progressivo esponenziale. Supera il limite presentato dalla media mobile semplice, in quanto tiene in maggior rilievo i valori recenti. Come per la SMA, i principali periodi utilizzati sono 50 e 200 [12].

**Convergenza e Divergenza di Medie Mobili (MACD).** Per costruire questo indicatore sono necessarie tre medie mobili esponenziali. Si calcola la differenza dei valori tra due medie mobili e la si utilizza per estrapolare una terza media mobile per generare segnali. Può essere usato per identificare gli aspetti del trend generale di un asset. Questo indicatore offre ottimi spunti nei casi in cui il trend (rialzista o ribassista) sia ben definito, può però causare falsi segnali nelle fasi di lateralizzazione. Una configurazione molto utilizzata è MACD(12, 26, 9), dove 12, 26, 9 corrispondono all'intervallo in cui operano le tre medie mobili [21].

**Indicatore di Forza Relativa (RSI).** Permette di misurare la velocità e la direzione

ne dei movimenti di prezzo. È considerato un indicatore di momentum (*tasso di salita o caduta del prezzo*), segnalando situazioni di ipercomprato e ipervenduto. Permette di anticipare eventuali inversioni di movimento di prezzo. Nel caso di un valore RSI inferiore a 30 si ha una situazione di ipervenduto, mentre con un RSI sopra 70 si è in ipercomprato [22].

**Indicatore Williams %R (%R).** È un indicatore di momentum che misura i livelli di ipercomprato e di ipervenduto di un titolo. Permette di individuare eventuali inversioni di trend. Il suo movimento è compreso tra zero e -100, indicando un valore di ipercomprato tra zero e -20 ed un valore di ipervenduto tra -80 e -100. Di solito il Williams %R è calcolato su un range temporale di 14 giorni, ma si possono utilizzare altri range temporali che meglio si adattano al movimento del titolo analizzato [51].

**Oscillatore Stocastico (KD).** Va ad individuare i massimi e i minimi del mercato in un determinato intervallo e li confronta con il prezzo attuale. Se il prezzo si avvicina al massimo storico si è in fase rialzista, nel caso opposto si è in fase ribassista. In altre parole, viene utilizzato per individuare tutte quelle situazioni di mercato che vengono definite come ipercomprato o ipervenduto. L'oscillatore stocastico ha un valore compreso tra 0 e 100. Di conseguenza se ci si trova al di sopra di 80, avremo una situazione di ipercomprato, se invece si è al di sotto di 20, la situazione è di ipervenduto. L'oscillatore stocastico, come la maggior parte degli strumenti dell'analisi tecnica, può essere modificato impostando i parametri più congrui alle proprie strategie di trading. Una classica configurazione per questo indicatore è su un intervallo di 14 giorni [34, 17].

## 3.2 L'analisi fondamentale nel caso del Bitcoin

Nei mercati finanziari tradizionali per analisi fondamentale, si intende un processo di raccolta e interpretazione di dati provenienti da una varietà di fonti diverse: notizie macroeconomiche, comunicati aziendali e bilanci. In questa analisi si presuppone che il prezzo di mercato di un bene non sia mai corretto. Come per l'analisi tecnica, nell'analisi fondamentale vige la soggettività degli analisti. Nel trading azionario l'analisi fondamentale viene utilizzata per analizzare il sentimento di mercato nei confronti di

un'azienda e determinare se un asset è sopravvalutato o sottovalutato, stabilendo-ne il "*valore intrinseco*", in modo da prendere decisioni che portino ad un profitto. Tuttavia, nei casi in cui si fa utilizzo di sistemi automatici per la compravendita nei mercati finanziari, risulta molto difficile integrare l'analisi fondamentale. Data la tipologia delle informazioni, perlopiù comunicati, notizie e bilanci, un software non è facilmente in grado di interpretare tali dati e agire di conseguenza. Risulta invece più facile sviluppare un bot per il trading che operi solo mediante analisi tecnica, in quanto, i dati sono rappresentati da valori numerici ed i concetti possono essere facilmente mappati in algoritmi decisionali.

I network di criptovalute non possono essere valutati attraverso la stessa lente usata per le compagnie tradizionali, nonostante alcuni indicatori come i sentimenti di mercato e notizie macroeconomiche possano essere utilizzati. In realtà, i sistemi più decentralizzati come Bitcoin (BTC) sono più vicini alle commodity (*materie prime*). Tuttavia, anche con le criptovalute più centralizzate (come quelle distribuite da organizzazioni), gli indicatori di analisi fondamentale tradizionali non ci dicono molto. Quindi, dobbiamo rivolgere l'attenzione a framework diversi. Il primo passo in questo processo è identificare metriche rilevanti [3]. Si tratta per la maggior parte di dati numerici, ricavati da analisi diretta sulla blockchain di ogni moneta. Questo tipo di informazioni può essere interpretato e mappato, da sistemi automatici, allo stesso modo degli indicatori tecnici finanziari visti precedentemente. Nello sviluppo di questa tesi, numerosi indicatori fondamentali sono stati utilizzati per allenare i modelli e, una descrizione completa di ogni feature è riportata nel Capitolo 6. Qui di seguito saranno presentate le categorie principali dei parametri di analisi fondamentale in criptovalute.

**Parametri on-chain.** Possono essere misurati esaminando i dati forniti dalla blockchain. I parametri principali sono: numero di transazioni, indirizzi attivi, commissioni pagate, hash rate (vedi Figura 3.1) e fondi in staking.

**Parametri del progetto.** Coinvolgono un approccio qualitativo che studia fattori come la performance del team (se ne esiste uno), il white paper e la futura roadmap.

**Parametri finanziari.** Sono parametri relativi al valore della rete stessa: capitalizzazione di mercato, offerta circolante, monete bruciate, liquidità, meccanismi di

offerta (*Stock-to-Flow*), offerta massima e tasso d'inflazione.

Se fatta correttamente, l'analisi fondamentale può fornire preziose informazioni sulle criptovalute, inaccessibili attraverso l'analisi tecnica. Riuscire a separare il prezzo di mercato dal valore "reale" di un network è un'abilità eccellente nel trading. Ovviamente, l'analisi tecnica ci svela cose che l'analisi fondamentale non può prevedere, per questo, molti trader usano una combinazione di entrambi gli approcci.

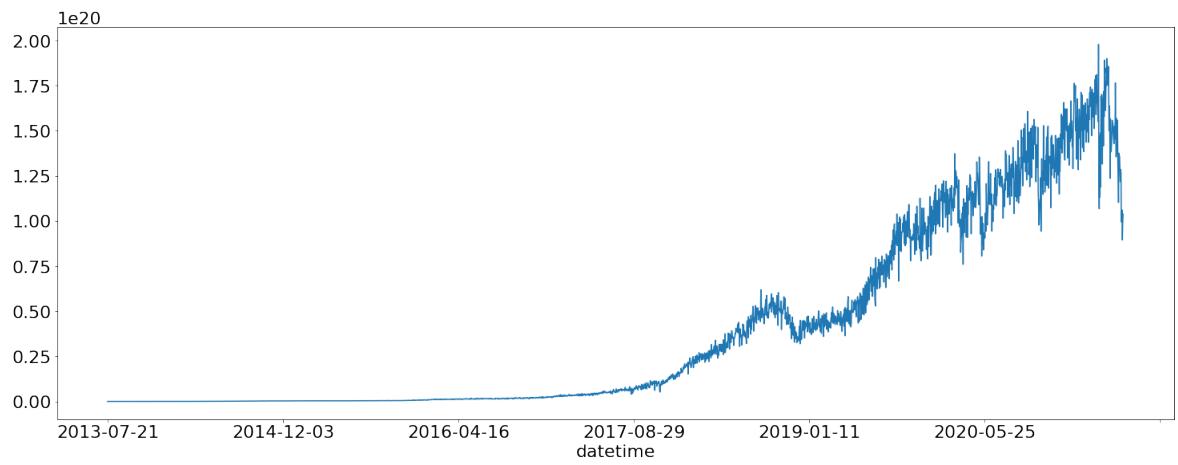


Figura 3.1: Bitcoin: Hash Rate, periodo: Luglio 2015/Giugno 2021, intervallo giornaliero.

# Capitolo 4

## Il Machine Learning

### 4.1 Introduzione al Machine Learning

Il Machine Learning ("*Apprendimento Automatico*") è una branca dell'intelligenza artificiale. I termini apprendimento automatico e intelligenza artificiale vengono spesso utilizzati, erroneamente, in modo interscambiabile. In realtà non hanno lo stesso significato, l'intelligenza artificiale rappresenta un grande insieme con al suo interno altre macro-categorie, susseguite da ulteriori sottoinsiemi più specifici (vedi Figura 4.1).

L'apprendimento è un qualsiasi processo che incrementa la performance del sistema attraverso l'**esperienza**. Per definizione quindi, l'apprendimento automatico è lo studio degli algoritmi che incrementano la loro **performance**, nello svolgimento di alcuni **task**, attraverso l'**esperienza**. Esempi di task in cui viene applicato il Machine Learning sono:

- **riconoscimento di pattern,**
- **ricognizione di anomalie,**
- **predizioni.**

L'ultimo punto dell'elenco rappresenta l'ambito principale dello studio di questa tesi, in cui si andrà ad analizzare il possibile impiego dell'apprendimento automatico nel campo della previsione degli andamenti dei prezzi futuri nei mercati. Nei paragra-

fi successivi di questo capitolo, saranno descritte le metodologie di apprendimento applicabili e i principali algoritmi, nonchè quelli utilizzati nella ricerca svolta.

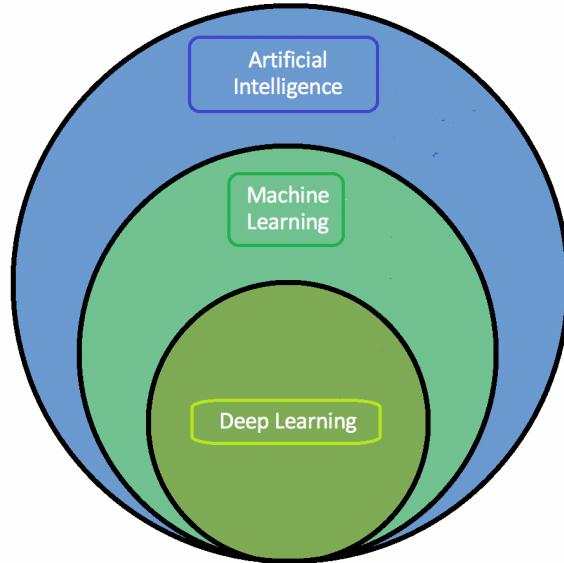


Figura 4.1: Artificial Intelligence, Machine Learning, Deep Learning

#### 4.1.1 I tipi di apprendimento

L’obiettivo principe dell’apprendimento automatico è che una macchina sia in grado di imparare dalla propria esperienza. In particolare una macchina dovrebbe assumere la capacità di portare a termine in maniera accurata compiti nuovi, mai affrontati prima, dopo aver fatto esperienza su un insieme di dati di addestramento. Si distinguono quattro tipologie di apprendimento, a seconda del processo e dei dati di cui l’algoritmo viene munito [60].

**Apprendimento supervisionato.** Il modello viene addestrato su un set di dati già etichettato. Le etichette (*labels*) rappresentano gli output desiderati. In questo modo l’algoritmo cerca di estrapolare una o più regole generali che associno l’input all’output corretto. I modelli basati su apprendimento supervisionato sono utilizzati in compiti di classificazione e regressione (vedi Sezione 4.1.2).

**Apprendimento non supervisionato.** Viene utilizzato un approccio più indipendente, nel set di dati di allenamento non vengono forniti gli output desiderati. Il modello impara a identificare processi e schemi complessi senza la guida attenta e costante fornita dalle etichette. Vengono utilizzati algoritmi di apprendimento non supervisionato principalmente in problemi di Clustering.

**Apprendimento semi-supervisionato.** Questo modello si trova a metà strada tra l'apprendimento supervisionato e quello non supervisionato. In fase di allenamento, di tutti i dati presenti nel training set, solo pochi di essi sono stati etichettati.

**Apprendimento rafforzativo.** Il modello lavora al fine di raggiungere un obiettivo. Nella fase di allenamento riceve ricompense in base alla sequenza di azioni svolte. Le ricompense svolgono il ruolo di riscontro, in questo modo l'algoritmo è in grado di capire se le azioni eseguite sono giuste o sbagliate. L'apprendimento per rinforzo è un processo iterativo a lungo termine.

#### 4.1.2 Classificazione e Regressione

Gli algoritmi di Machine Learning ad apprendimento supervisionato sono in grado di affrontare due tipologie di compiti: *classificazione e regressione*. Entrambe le tecniche permettono di prevedere il valore di un particolare attributo basandosi sul valore di altri attributi (task predittivi).

**Classificazione.** È il task di apprendere una "funzione target" in grado di mappare ogni set di attributi in una classe predefinita. I dati di input per un classificatore sono collezioni di *record*. Ogni record è caratterizzato da una tupla  $(x, y)$ , dove  $x$  è un set di attributi e  $y$ , detto **attributo speciale**, rappresenta l'etichetta di classe (categoria). Le etichette sono valori discreti non ordinati, caratteristica chiave che contraddistingue un classificatore da un regressore, in cui le etichette  $y$  sono valori continui.

Più formalmente:

*Date delle coppie di valori  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , un modello di classificazione ha l'obiettivo di costruire una funzione  $f(x)$  in grado di prevedere un'etichetta  $y$  dato  $x$ .*

Per addestrare un modello di classificazione si forniscono set di attributi già etichettati, in questo modo l'algoritmo apprende schemi e correlazioni tra i dati  $x$  e le classi  $y$ . Un modello addestrato dovrà quindi essere in grado di associare dei record  $x$ , mai visti prima, ad un classe  $y$ . La classificazione può essere lineare o a più dimensioni (vedi Figura 4.2).

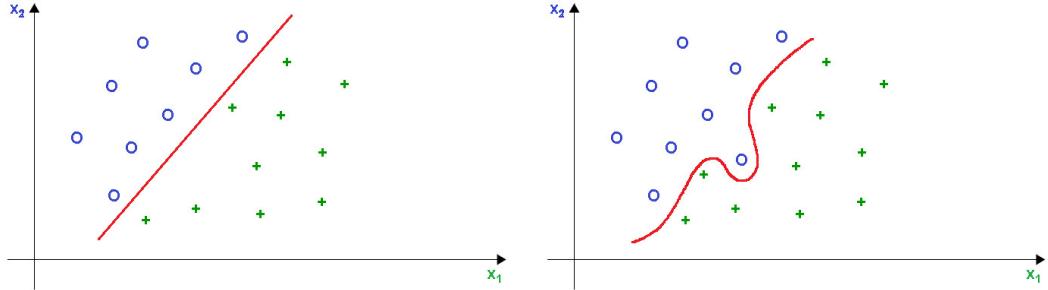


Figura 4.2: Esempi di classificazione lineare (*sinistra*) e classificazione a più dimensioni (*destra*).

**Regressione.** Consiste nello stimare un valore reale (continuo), mentre la classificazione genera un valore discreto (classe). Allo stesso modo del classificatore, i dati di input sono collezioni di *record* costituiti da tuple  $(x, y)$ . Gli algoritmi di regressione hanno l'obiettivo di determinare una funzione che sia in grado di calcolare, con la più elevata precisione possibile, un valore  $y$  dato un set di attributi  $x$ . L'output di un regressore, invece che una valore discreto, corrisponde ad un valore continuo.

Più formalmente:

*Date delle coppie di valori  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , un modello di regressione ha l'obiettivo di costruire una funzione  $f(x)$  in grado di prevedere un valore continuo  $y$  dato  $x$ .*

Per riportare un caso specifico, un classificatore è in grado di prevedere e determinare se il prezzo futuro di Bitcoin salirà o scenderà, fornendo in output una classe a seconda della previsione (ad esempio "Up" e "Down"). Un regressore, a sua volta, sarà in grado di restituire un valore di prezzo (ad esempio "45'000.00\$") e non un'etichetta categorica.

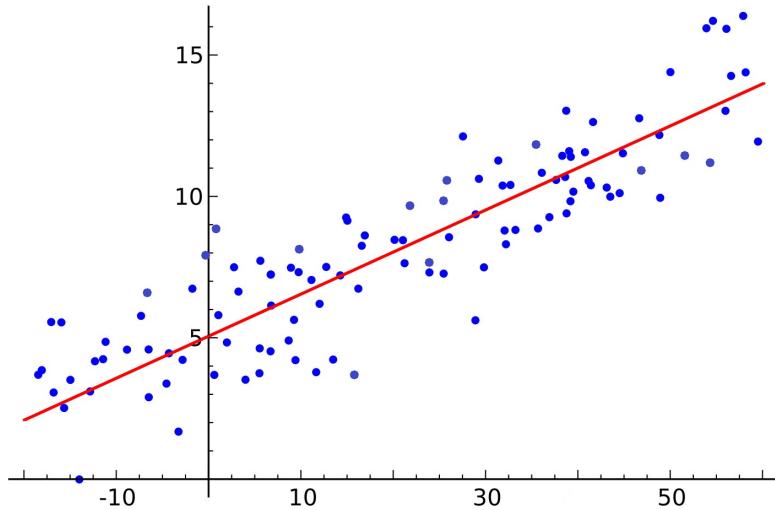


Figura 4.3: Esempio di regressione.

## 4.2 Random Forest

Una **foresta casuale** (*random forest*) è un metodo di apprendimento d'insieme per problemi di classificazione e regressione. Una foresta casuale è **costituita da una moltitudine di alberi di decisione** (vedi Sezione 4.2.1). Il primo algoritmo per foreste *decisionali* casuali fu ideato nel 1995 da Tin Kam Ho [36] utilizzando il metodo del *sottocampionamento casuale*, chiamato anche **attribute bagging** o **feature bagging**. Un'estensione dell'algoritmo originario fu sviluppata da Leo Breiman [10], il quale integrò la propria idea di "bagging" (vedi Sezione 4.2.2) a quella del sottocampionamento casuale precedentemente proposta da Tin Kam Ho.

### 4.2.1 Alberi decisionali

In teoria dei grafi, un albero è un grafo connesso e privo di cicli. Un *albero decisionale* (**decision tree**) è un sistema gerarchico che consiste in nodi e archi direzionali. I nodi sono distinti in: **nodi radice, nodi interni (o intermedi) e nodi foglia (o terminali)**. Ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e, una foglia rappresenta il valore predetto per la variabile obiettivo. Le variabili possono essere discrete o continue, a seconda del caso sia parlerà rispettivamente di classificazione o regressione. Il processo

dei dati ricevuti in input da un albero decisionale è simile ad una serie di test/domande (rappresentate dai nodi interni), le quali permettono di scorrere attraverso la gerarchia dell’albero, in uno dei suoi percorsi, fino ad arrivare ad un nodo foglia che costituirà la predizione ( $y$ ) associata all’input ( $x$ ) (vedi Figura 4.2.1).

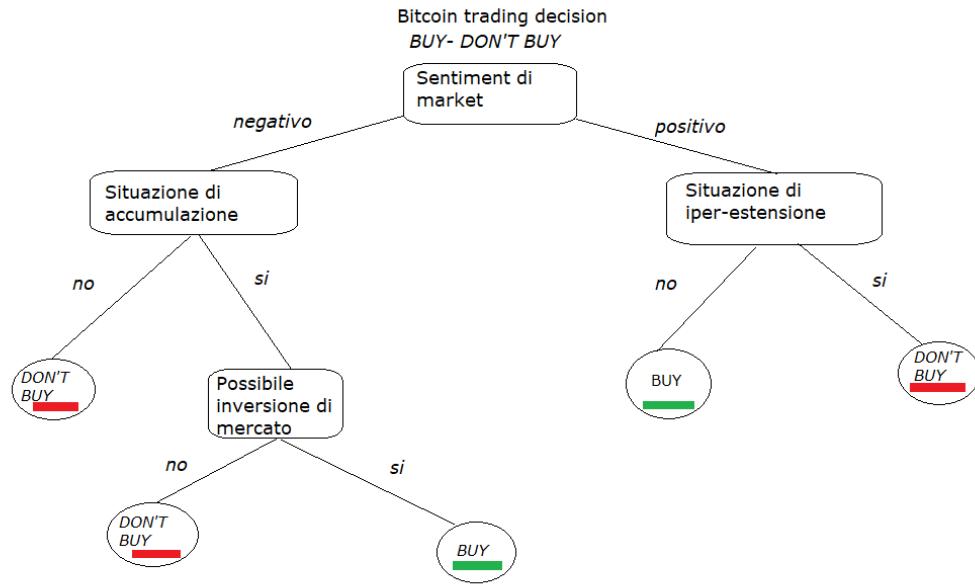


Figura 4.4: Esempio di albero decisionale binario con variabili discrete.

La base fondamentale, di un algoritmo decision tree, sta nel costruire un albero in grado di operare con la miglior precisione possibile. Il numero di alberi decisionali che può essere costruito da un set di attributi è di ordine esponenziale. Alcuni alberi sono più precisi di altri, tuttavia costruirne uno **ottimo** risulta infattibile. Ciò è dovuto all’enorme grandezza dello spazio di ricerca. In aiuto, sono stati implementati algoritmi in grado di trovare alberi decisionali molto accurati, in un lasso di tempo ragionevole. Questi algoritmi adoperano solitamente una strategia **greedy**, un esempio è l’*Algoritmo di Hunt*. In questa tecnica l’albero di decisione viene costruito in modo ricorsivo, partizionando i record di training in sottoinsiemi sempre più puri. La suddivisione è permessa dalle **test condition**: condizioni sugli attributi dei dati di allenamento che permettono di distinguerli. Le condizioni di test vengono effettuate ad ogni passo della creazione dell’albero. Questo processo continua fino a

raggiungere il punto in cui tutti i record di un sottoinsieme possono essere associati allo stesso valore (non sono necessarie ulteriori sottodivisioni, si è in un nodo *foglia*). Numerosi ulteriori algoritmi, a partire dall'implementazione di Hunt, sono stati definiti presentando migliore applicabilità, efficacia e versatilità. Un buon algoritmo per determinare alberi decisionali deve cercare di effettuare *test condition* con la finalità di suddividere i record in *split* (sottoinsiemi) più piccoli possibile ad ogni passo. La scelta dello split migliore può esser fatta attraverso varie metriche, a seconda se si sta affrontando una caso di classificazione o regressione. Gli algoritmi per la costruzione di alberi decisionali funzionano dall'alto verso il basso, scegliendo una variabile in ogni passaggio che suddivide al meglio l'insieme di elementi [58].

In un **problema di classificazione**, viene presa in considerazione la distribuzione dei record, prima e dopo lo split. Sia  $p(i/t)$  la frazione di record appartenenti alla classe  $i$  in un nodo  $t$ . In un problema con due classi, la distribuzione in classi può essere scritta come  $(p_0, p_1)$  ad ogni nodo, dove  $p_1 = 1 - p_0$ . La distribuzione  $p$  viene utilizzata come parametro per calcolare il **grado di impurità** dei nodi a seguito di un eventuale split, con l'intento di determinarne la bontà. Ad esempio, uno split in un nodo che genera una distribuzione di classe  $(0, 1)$  ha impurità uguale a 0, mentre uno con  $(0.5, 0.5)$  ha l'impurità più alta.

Le principali metriche di misurazione per l'impurità sono:

- **Entropia**

$$I_H(t) = - \sum_{i=0}^{c-1} p(i/t) \log_2 p(i/t)$$

- **Gini**

$$I_G(t) = 1 - \sum_{i=0}^{c-1} [p(i/t)]^2$$

- **Errore di classificazione**

$$I_E(t) = 1 - \max_i [p(i/t)]$$

dove  $c$  è il numero di classi.

Tali indicatori d'impurità hanno valore massimo quando i record sono equamente distribuiti tra le classi e, hanno valore minimo quando tutti i record appartengono

ad una sola classe. Per determinare la qualità di una test condition è necessario confrontare il **grado di impurità del nodo padre** con il **grado di impurità del nodo figlio**: maggiore sarà la differenza e meglio avrà agito la condizione di test. Il **guadagno (gain)** è un criterio che può essere utilizzato per determinare la bontà dello split. Viene indicato con  $\Delta$  ed è dato dalla formula:

$$\Delta = I_{parent} - \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j)$$

dove:

- $I$  è l'impurità di un dato nodo;
- $N$  è il numero totale di record al nodo padre;
- $k$  è il numero di valori dell'attributo;
- $N(v_j)$  è il numero di record associati al nodo figlio  $v_j$ .

Gli algoritmi per gli alberi di decisione cercheranno di massimizzare il **gain**, il che risulta equivalente a minimizzare l'impurità media dei nodi figli.

In **problemi di regressione**, per determinare lo split migliore, sono utilizzate metriche per l'impurità adatte a variabili continue. Questo perché l'uso delle metriche precedenti richiederebbe la discretizzazione prima di essere applicata (come nel caso dell'utilizzo di  $Gini(t)$ ). La misura di impurità può essere calcolata mediante la tecnica di **riduzione della varianza**, utilizzata per determinare l'omogeneità di un nodo. È così chiamata perché utilizza la varianza come misura per decidere la caratteristica su cui suddividere un nodo in nodi figli. Il criterio che genera lo split con la varianza minore viene selezionato. Se un nodo è completamente omogeneo, la varianza è zero.

$$Varianza = \frac{\sum(X - \bar{X})^2}{N}$$

dove:

- $X$  è il valore attuale;
- $\bar{X}$  è la media dei valori;

- $n$  è il numero di valori;

Gli alberi di decisione si adattano sia molto bene che velocemente ai dati di addestramento e, possono gestire facilmente attributi ridondanti o irrilevanti. Purtroppo la dimensione di quest'ultimi può essere esponenziale, nonostante algoritmi accurati, se non limitati, soffrono in maniera sistematica del problema di *Overfitting* (vedi Sezione 4.5).

### 4.2.2 Bagging

**Bagging** è l'abbreviazione di **Bootstrap Aggregation**.

Il *bootstrap* è un metodo statistico di ricampionamento con reimmissione, utilizzato per stimare metriche come varianza e media di una popolazione di dati. Viene spesso utilizzato nell'apprendimento automatico per stimare le performance dei modelli. In questo approccio si effettua un sottocampionamento casuale del dataset reinserendo nell'insieme originale anche i record già selezionati. Così ogni record ha la stessa probabilità di essere ripescato per un campione.

Nella fase di costruzione degli alberi di decisione per una foresta casuale, il bootstrap è utilizzato per **suddividere il set di dati di allenamento in sottocampioni casuali**. Gli alberi decisionali sono sensibili ai dati di allenamento su cui vengono costruiti. Se i dati di allenamento vengono modificati (ad es. un albero viene addestrato su un sottoinsieme del dataset) l'albero decisionale risultante è diverso e, a loro volta, le previsioni possono essere piuttosto diverse.

Nel **bagging** per foreste casuali viene utilizzato lo stesso metodo tramite una procedura di *ensemble*. *Un metodo ensemble è una tecnica che combina le previsioni di più algoritmi di apprendimento automatico per fare previsioni più accurate rispetto a qualsiasi singolo modello* [28].

Il bagging rappresenta una procedura generale che può essere utilizzata negli algoritmi ad alta varianza, come nel caso di alberi decisionali.

### 4.2.3 Dal Bagging alle Random Forests

Una **foresta casuale** *combina molti alberi decisionali in un unico modello* (vedi Figura 4.5). Oltre alla tecnica di bagging sul sottocampionamento del dataset di allenamento (in fase di costruzione degli alberi), viene utilizzata un'altra tipologia di

schema di bagging per l’implementazione di una foresta. Durante la fase di divisione in split, per la creazione dei nodi dell’albero, l’algoritmo di apprendimento utilizza sottoinsiemi casuali di funzionalità (***feature bagging***) da considerare per effettuare le *test condition*. Ogni albero della foresta opererà con campioni di funzionalità e campioni di dati casuali, così, addestrando ciascun albero su campioni diversi, nel complesso, l’intera foresta avrà una varianza inferiore. L’idea generale di una tecnica di bagging è che *la combinazione di più modelli di apprendimento incrementi il risultato complessivo*. Individualmente, le previsioni fatte dagli alberi decisionali potrebbero non essere accurate ma, combinate insieme, le previsioni saranno in media più vicine al risultato [28].

*Il risultato finale restituito da una foresta casuale non è altro che la media del risultato numerico restituito dai diversi alberi (nel caso di un problema di regressione), o la classe restituita dal maggior numero di alberi (nel caso la random forest sia stata utilizzata per risolvere un problema di classificazione).*

Per quanto riguarda problemi come l’overfitting, a differenza degli alberi decisionali, una foresta casuale è più resistente, in quanto, grazie al sottocampionamento casuale, gli alberi costruiti sono meno profondi. Ovviamente il costo computazionale per costruire una random forest è superiore a quello di costruire un decision tree, ma i vantaggi nel farlo sono maggiori. Una previsione più accurata richiede più alberi, il che si traduce in un modello più lento.

Un’altra grande qualità degli algoritmi di foresta casuale sta nel poter determinare con facilità **l’importanza relativa di ciascuna feature nella previsione** (vedi Sezione 6.5). Osservando l’importanza di ciascuna feature, è possibile decidere quale eliminare tra quelle che non contribuiscono abbastanza al processo di previsione (vedi Sezione 6.2.1). Diminuendo il numero di feature si riduce il rischio di overfitting.

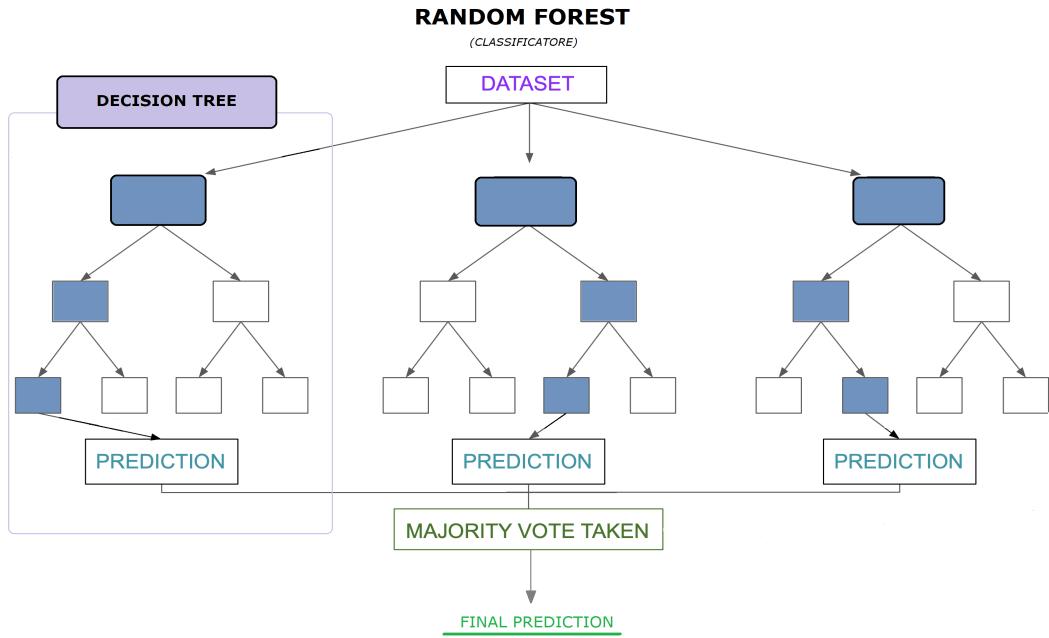


Figura 4.5: Esempio del funzionamento di un modello basato su foresta casuale per problemi di classificazione.

È possibile implementare dei modelli per classificazione e regressione, basati su foreste casuali, sfruttando la libreria Python ***scikit-learn*** [55]. Una varietà di iperparametri può essere perfezionata a seconda dell'utilizzo. Gli iperparametri nella foresta casuale vengono utilizzati per aumentare la performance predittiva del modello o, per rendere il modello più veloce. Tra i più importanti vi sono;

- **n\_estimators:** corrisponde al numero di alberi nella foresta. Un numero maggiore di alberi tende a migliorare e a rendere le previsioni più stabili, tuttavia ne rallenta la costruzione e il calcolo.
- **max\_feature:** corrisponde al numero massimo di feature che la foresta casuale considera per dividere un nodo.
- **min\_samples\_split:** determina il numero minimo di elementi richiesti per effettuare uno split su un nodo interno.
- **min\_samples\_leaf:** determina il numero minimo di elementi richiesti per avere un nodo foglia.

- **max\_depth**: corrisponde alla profondità massima raggiungibile da ogni albero della foresta. Se non viene definito, i nodi vengono espansi finché tutte le foglie non sono pure o, finché tutte le foglie non contengono meno di *min\_samples\_split* elementi.

Le random forest vengono utilizzate in numerosi ambiti, come predizioni sui mercati azionari, in ambito medico e bancario. In finanza, ad esempio, viene utilizzato per rilevare i clienti che hanno maggiori probabilità di ripagare il debito in tempo o utilizzare più frequentemente i servizi di una banca, oppure può essere utilizzato per determinare il comportamento futuro di un'azione.

## 4.3 Support Vector Machine

In apprendimento automatico, per **Support Vector Machine** (anche nota come **Macchina a vettori di supporto** o **SVM**) si intende un modello di apprendimento supervisionato, la cui implementazione permette di affrontare problemi di classificazione e regressione. La prima ideazione fu di Vladimir Vapnik, durante gli anni sessanta, in cui veniva proposto un modello in grado di risolvere problemi di sola classificazione lineare. Nel 1992 una rivisitazione dello stesso Vapnik e dei suoi colleghi ricercatori, estese il problema a classificazioni non lineari [9].

### 4.3.1 Classificazione

L'idea alla base di una SVM, in problemi di classificazione, è *di trovare un iperpiano di separazione per le classi, che massimizzi il margine tra le classi stesse, dove per margine si intende la distanza minima dalla retta ai punti del dataset*. I punti sono:

- i record del dataset di allenamento nel momento in cui si sta costruendo il modello;
- i dati di test nel momento in cui la macchina a vettori di supporto sta classificando record mai visti prima.

**Iperpiano.** Per un'attività di classificazione binaria, l'iperpiano è rappresentato da una retta che separa due classi (vedi Figura 4.6). A 3 dimensioni, un iperpiano è

rappresentato da un piano. Con più di tre dimensioni la rappresentazione è data da figure più complesse, a seconda della dimensione problema.

**Vettori di Supporto.** Sono i punti del dataset di allenamento, l'obiettivo dell'algoritmo è di trovare quelli che permettano di costruire il miglior iperpiano. Sono considerati gli elementi critici di un dataset, influenzano tutti gli altri dati ai fini della classificazione. Infatti, i vettori di supporto, definiscono la posizione dell'iperpiano corrispondente, se i punti vengono rimossi o modificati quest'ultima cambia. Come nel caso della Figura 4.6, i vettori di supporto sono i due punti (uno per classe) più vicini alla retta che rappresenta l'iperpiano. Nella determinazione di questi punti, l'algoritmo cerca di trovare quelli che ne massimizzano il margine.

**Margine.** È definito come la distanza tra i vettori di supporto, relativi a classi differenti, più vicini. A metà di tale distanza viene tracciato l'iperpiano.

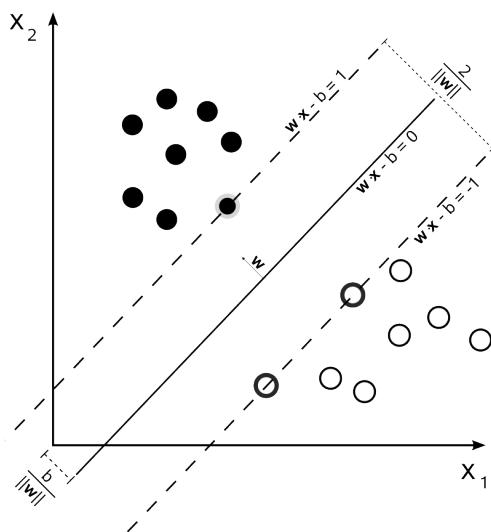


Figura 4.6: Esempio di separazione lineare di una SVM [64].

Un algoritmo per SVM, nel definire un iperpiano, cerca i vettori di supporto più vicini all'altra classe e, allo stesso tempo cerca di massimizzare la distanza tra i due. Un iperpiano che rispetti a pieno tali caratteristiche sarà considerato *ottimale*, ossia generi il minor errore di classificazione su una nuova previsione. Questo perché più lontano dall'iperpiano si trovano i punti da analizzare, più si è fiduciosi che la classificazione

avvenga in modo corretto.

**Classificazione Lineare.** Nel caso di un problema di classificazione, ogni iperpiano può essere rappresentato con la seguente formula:

$$\bar{w}x - b = 0$$

dove:

- $\bar{w}$  è il vettore di peso;
- $x$  è il vettore di caratteristiche di input (record del dataset di allenamento);
- $\frac{b}{\|\bar{w}\|}$  rappresenta la traslazione dell'iperpiano rispetto all'origine, lungo il vettore  $\bar{w}$ .

In  $n$  dimensioni l'iperpiano è dato dalla combinazione lineare di tutte le dimensioni uguagliate a zero. La formula precedente può anche essere scritta in modo più dettagliato:

$$w_1x_1 + w_2x_2 + \dots + w_nx_n - b = 0.$$

Ad esempio, in un problema a due dimensioni si avrebbe:

$$w_1x_1 + w_2x_2 - b = 0.$$

Qualsiasi punto al di sopra e al di sotto dell'iperpiano rispetta rispettivamente le seguenti disequazioni:

$$\bar{w}x - b > 0$$

$$\bar{w}x - b < 0.$$

**In un problema di classificazione linearmente separabile**, è possibile selezionare due iperpiani paralleli, che separano i dati in due classi, in modo da avere la distanza tra i due (margine) maggiore possibile. Le equazioni corrispondenti sono:

$$\bar{w}x - b = 1,$$

in cui si evince che qualsiasi punto al di sopra di tale iperpiano sarà etichettato con 1 e,

$$\bar{w}x - b = -1.$$

in cui si evince che qualsiasi punto al di sotto di tale iperpiano sarà etichettato con -1. La distanza tra i due confini, rappresenta il margine e corrisponde a  $\frac{2}{\|w\|}$ , dove il denominatore corrisponde alla lunghezza o "norma" del vettore  $\bar{w}$ . L'obiettivo dell'algoritmo SVM è massimizzare il margine, trovando un valore più piccolo possibile per  $\|w\|$ .

Dalle equazioni precedenti si può in constatare che ogni record  $i$  del dataset di allenamento rispetta una delle due seguenti equazioni:

$$\bar{w}x_i - b \geq 1, \text{ se } y_i = 1.$$

(con  $y_i$  etichetta di classe), o

$$\bar{w}x_i - b \leq -1, \text{ se } y_i = -1.$$

Le precedenti equazioni possono essere racchiuse in:

$$y_i(\bar{w}x_i - b) \geq 1, \text{ per ogni } 1 \leq i \leq n.$$

Il problema di ottimizzazione può essere così formulato:

$$\text{Minimizzare } \|w\| \text{ in } y_i(\bar{w}x_i - b) \geq 1 \text{ per } i = 1, \dots, n.$$

**Classificazione non Lineare.** Nel 1992 V. Vapnik, insieme a B. Boser e I. Guyon, mediante l'utilizzo del **metodo Kernel** [9], presentò una tecnica per implementare un algoritmo SVM in grado di affrontare problemi di classificazione non lineare. L'algoritmo è simile al caso delle classi separabili linearmente, se non per il fatto che ogni prodotto scalare viene sostituito con una **funzione kernel non lineare**.

**Kernel.** Alcuni algoritmi di apprendimento automatico, come SVM, utilizzano un insieme di funzioni matematiche definite come kernel. Lo scopo è di prendere i dati in input e, nel caso in cui non sia possibile determinare un iperpiano in grado di separare linearmente tali dati, trasformarli nella forma richiesta (vedi Figura 4.7). Il Kernel può essere definito come:

$$K(x, y) = \langle f(x), f(y) \rangle$$

dove  $K$  è la **funzione del Kernel**,  $x$  e  $y$  sono vettori di input di dimensione  $n$ . La funzione  $f$  viene utilizzata per mappare l'input dallo spazio  $n$  dimensionale a quello di livello più alto,  $m$  dimensionale. È possibile utilizzare varie tipologie di funzione kernel, qui di seguito alcune tra le più diffuse:

- **Kernel lineare:**

$$K(x_i, y_j) = x_i \cdot y_j$$

- **Kernel polinomiale:**

$$K(x_i, y_j) = (x_i \cdot y_j + c)^d$$

dove  $c$  è una costante e  $d$  un grado di libertà. Un valore di 1 per  $d$  rappresenta il kernel lineare. Un valore maggiore rende il limite decisionale più complesso e potrebbe comportare un problema di overfitting (vedi Sezione 4.5).

- **Kernel RBF (Radial Basis Function):**

$$K(x_i, y_j) = e^{(-\gamma \|x_i - y_j\|^2)}$$

chiamato anche Kernel gaussiano. Contiene un parametro  $\gamma$  che, assumendo un valore piccolo, fa sì che l'algoritmo si comporti come un SVM lineare, mentre un valore grande rende il modello fortemente influenzato dai vettori di supporto.

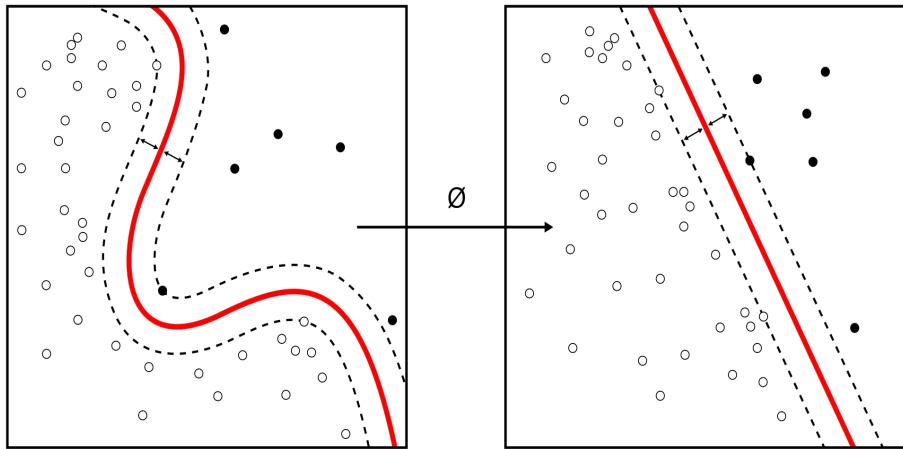


Figura 4.7: Esempio di trasformazione di un problema non linearmente separabile in un problema linearmente separabile, mediante l'utilizzo di una funzione Kernel [65].

### 4.3.2 Regressione

Le macchine a vettori di supporto possono essere utilizzate anche per problemi di regressione, mantenendo tutte le caratteristiche principali che caratterizzano l'algoritmo (margin massimo). La versione di SVM per problemi di regressione è stata proposta nel 1996 da Vladimir Vapnik, Harris Drucker, Christopher J. C. Burges, Linda Kaufman e Alexander J. Smola [20]. Questo metodo è chiamato **Support Vector Regression** (SVR), e rappresenta un algoritmo di apprendimento supervisionato che utilizza gli stessi principi della SVM per la classificazione, con solo alcune piccole differenze. Come nell'algoritmo di classificazione (in cui nella fase di costruzione del modello l'algoritmo non si preoccupa dei punti di addestramento che si trovano oltre il margine), il modello prodotto da SVR dipende soltanto da un sottoinsieme dei dati di addestramento. L'algoritmo di costruzione del modello ignora qualsiasi dato di addestramento vicino alla previsione del modello. A differenza di altri modelli di regressione che cercano di ridurre al minimo l'errore di previsione (differenza tra predizione e valore reale), SVR cerca di adattare l'iperpiano entro un valore di soglia  $\varepsilon$  (margine di tolleranza): tutte le previsioni devono rientrare in un intervallo  $\varepsilon$  [62]. Il valore di soglia è la distanza tra l'iperpiano e il confine del margine (vedi Figura 4.8). L'idea principale quindi è sempre la stessa: minimizzare l'errore, individuando l'iperpiano che massimizza il margine, tenendo presente che una parte dell'errore è

tollerata.

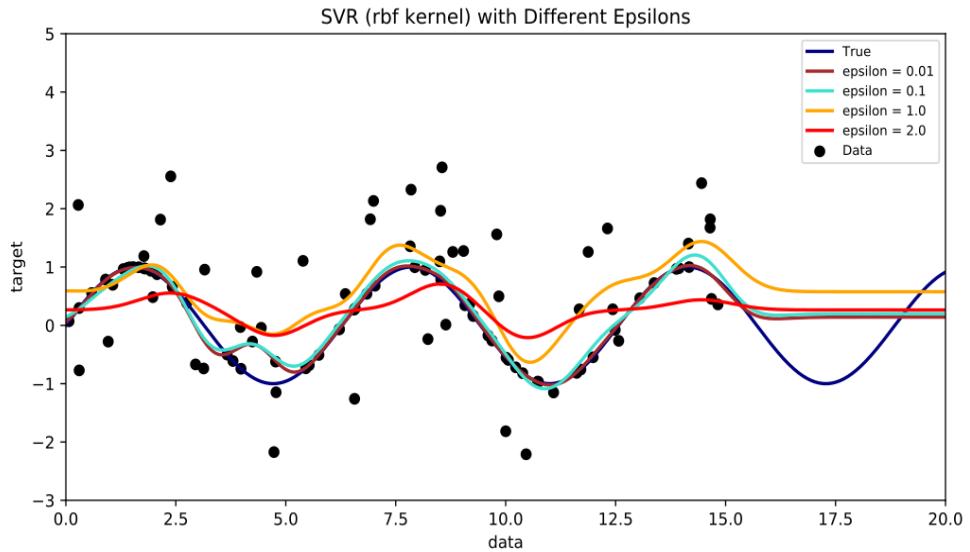


Figura 4.8: SVR con diverse soglie  $\varepsilon$ . All'aumentare di  $\varepsilon$ , la previsione diventa meno sensibile agli errori [66].

Come per le Random Forests, è possibile implementare modelli SVM sfruttando **scikit-learn** [55]. I modelli implementati, per la ricerca effettuata in questa tesi, sono:

- **SVC** per la classificazione;
- **SVR** per la regressione.

Ovviamente, è possibile impostare vari parametri per ottimizzare il modello a seconda del problema e dei dati su cui deve lavorare. Ad esempio è possibile utilizzare differenti funzioni kernel, distinguendo tra: lineare, polinomiale e RBF. Nel caso i problemi da affrontare siano di tipo lineare, scikit-learn predispone anche di modelli più veloci, utilizzabili esclusivamente in problemi di questo tipo. Tra gli approcci più comuni nell'utilizzo di modelli SVM vi sono: categorizzazione di testo [42], analisi semantica [57], classificazione di immagini [5].

## 4.4 Deep Learning e Artificial Neural Network

L'**Apprendimento Profondo** (in inglese *Deep Learning*) è una branca del Machine Learning. È basato su Reti Neurali Artificiali (vedi Sezione 4.4.2). La parola "profondo" deriva dal fatto che, nel Deep Learning, si utilizzano più livelli successivi, con il compito di estrarre progressivamente funzionalità di livello superiore, a partire dall'input non elaborato [19]. Ad esempio, nell'elaborazione delle immagini, i livelli inferiori possono identificare linee e bordi, mentre i livelli superiori possono identificare figure più complesse come cifre, lettere e volti.

Un algoritmo di apprendimento profondo è in grado di apprendere da solo quali caratteristiche considerare per ciascun livello e a quali feature dare maggior importanza, al fine di ottimizzare il modello predittivo [6]. Ciò non elimina completamente la necessità di un individuo esperto nella fase di progettazione, è infatti necessario stabilire con precisione i vari parametri di un modello, come il numero di livelli, le funzioni di attivazione e il numero di neuroni (vedi Sezione 4.4.3).

La grande potenza del Deep Learning permette di essere utilizzato in svariati ambiti, dal riconoscimento vocale, all'analisi di immagini mediche, fino alla Computer Vision [38, 13, 43]. Nei sottoparagrafi seguenti saranno descritti i principali concetti sulle Reti Neurali Artificiali, da modelli a singolo strato, fino alle reti multi-livello. Sarà poi descritto un particolare modello di rete, utilizzato nel caso specifico di questa ricerca.

### 4.4.1 Perceptron

Le **Reti Neurali Artificiali** sono modelli predittivi ispirati alle reti neurali biologiche, le quali costituiscono il cervello degli animali. Analogamente alla struttura del cervello umano, una **ANN** (acronimo di *Artificial Neural Network*), si basa su un interconnesso sistema di nodi e link (archi) direzionali. I nodi sono chiamati **neuroni artificiali**, mentre ogni connessione, come le sinapsi in un cervello biologico, ha lo scopo di trasmettere un segnale tra i neuroni.

Un primo esempio, nonchè il più semplice, è il modello **Perceptron** (*Percettrone* in italiano). Fu proposto per la prima volta nel 1958 dallo psicologo Frank Rosenblatt [59]. Il Percettrone era un semplice classificatore lineare binario, in grado di appren-

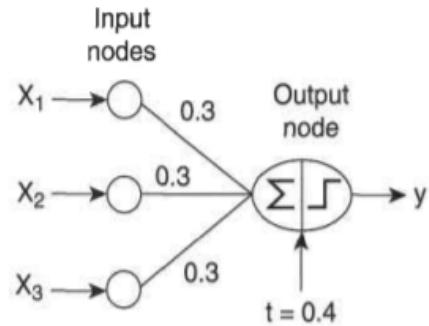
dere efficacemente la regola necessaria per riconoscere due classi di input diverse e linearmente separabili. Un percettrone consiste di due tipi di nodi:

- **nodi di input;**
- **nodi di output.**

I nodi di input sono collegati al nodo di output mediante un **arco pesato**. Il collegamento pesato viene utilizzato per emulare la forza della connessione sinaptica tra i neuroni. La **fase di allenamento**, di un modello perceptron, consiste nell'adattare i pesi dei collegamenti, in modo da mappare le relazioni *input-output* dei dati sottostanti. In Figura 4.9 è mostrato un dataset con 3 variabili booleane ( $x_1, x_2, x_3$ ) e una variabile di output  $y$ . Quest'ultima, assume rispettivamente un valore di  $-1$  se almeno due dei tre input sono uguali a zero, altrimenti assume il valore  $+1$  se almeno due degli input sono maggiori di zero [68]. La semplice architettura in Figura 4.9 (b) rappresenta il percettrone, in cui abbiamo i nodi di input, ognuno dei quali collegato con un arco pesato al nodo di output.

$X_1$	$X_2$	$X_3$	$y$
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

(a) Data set.



(b) Perceptron.

Figura 4.9: Esempio di funzionamento di un modello Perceptron [68].

Un percettrone calcola il suo valore di output  $\bar{y}$  eseguendo una somma ponderata sui dati in input, sottraendo un fattore di polarizzazione  $t$  (**bias**) dalla somma, e quindi esaminando il **segno** del risultato. Seguendo questa definizione, nell'esempio

in Figura 4.9, l'output  $\bar{y}$  è così calcolato:

$$f(x) = \begin{cases} 1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0; \\ -1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0. \end{cases}$$

Il nodo di input, trasmette il valore ricevuto al collegamento in uscita senza eseguire nessuna trasformazione su di esso. Il nodo di output, invece, è un dispositivo matematico che calcola la somma ponderata dei suoi input, sottrae il *bias* e quindi produce un output, che dipenderà dal segno della somma risultante. Il bias rappresenta la **soglia di attivazione** (valore per il quale il neurone si attiva). Più specificamente, l'output di un modello perceptron può essere espresso matematicamente come segue:

$$\bar{y} = \text{sign}(w_d x_d + w_{d-1} x_{d-1} + \cdots + w_2 x_2 + w_1 x_1 - t),$$

dove  $w_1, w_2, \dots, w_d$  sono i pesi dei collegamenti di input, e  $x_1, x_2, \dots, x_d$  sono i valori degli attributi di input. La funzione segno, funge da **funzione di attivazione** per il neurone di output: ritorna  $+1$  se l'argomento è positivo, ritorna  $-1$  se l'argomento è negativo. Il modello perceptron può essere scritto in una forma più compatta come segue

$$\bar{y} = \text{sign}[w_d x_d + w_{d-1} x_{d-1} + \cdots + w_1 x_1 + w_0 x_0] = \text{sign}(\mathbf{w} \cdot \mathbf{x}),$$

dove  $w_0 = -t$ ,  $x_0 = 1$ , e  $\mathbf{w} \cdot \mathbf{x}$  è il prodotto scalare tra il vettore dei pesi  $\mathbf{w}$  e il vettore degli attributi di input  $\mathbf{x}$ .

**Apprendimento del modello Perceptron.** Durante la fase di allenamento, di un perceptron, i parametri di peso  $w$  vengono regolati fino a quando gli output del modello diventano coerenti con le vere uscite degli esempi di training (etichette dei dati di allenamento). I passaggi svolti in fase di allenamento sono presentati nell'Algoritmo 1.

---

**Algorithm 1** Algoritmo di apprendimento Perceptron [68].

---

Sia  $D = \{(x_i, y_i) | i = 1, 2, \dots, N\}$  l'insieme dei dati di allenamento

Inizializzo il vettore dei pesi  $w^{(0)}$  con valori casuali

**repeat**

**for** ogni record di training  $(x_i, y_i) \in D$  **do**

Calcola l'output  $\bar{y}_i^{(k)}$

**for** ogni peso  $w_j$  **do**

Aggiorna il peso,  $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \bar{y}_i^{(k)})x_{ij}$ .

**end for**

**end for**

**until** la condizione di arresto viene soddisfatta

---

Il calcolo chiave dell'algoritmo è la formula di aggiornamento del peso:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \bar{y}_i^{(k)})x_{ij}$$

dove:

- $w_j^{(k+1)}$  è il parametro di peso, associato con l'i-esimo collegamento di input dopo la j-esima iterazione;
- $\lambda$  è un parametro noto come **learning rate** (tasso di apprendimento);
- $x_{ij}$  è il valore del j-esimo attributo del record di training  $x_i$ .

Il nuovo peso  $w^{(k+1)}$  è una combinazione del vecchio peso  $w^{(k)}$ , e un termine proporzionale all'*errore di previsione* ( $y - \bar{y}$ ). Se la previsione è corretta il peso rimane invariato, in caso contrario viene modificato nei seguenti modi:

- se  $y = +1$  e  $\bar{y} = -1$ , allora l'errore di previsione è  $(y - \bar{y}) = 2$ . Per compensare all'errore, viene incrementato il valore dell'output previsto, aumentando i pesi di tutti i collegamenti con input positivi e, diminuendo tutti i pesi dei collegamenti con input negativi;
- se  $y_i = -1$  e  $\bar{y} = +1$ , allora  $(y - \bar{y}) = -2$ . Per compensare all'errore, viene decrementato il valore dell'output previsto, diminuendo i pesi di tutti i collegamenti e, aumentando i pesi di tutti i collegamenti negativi.

Nella formula di aggiornamenti del peso, i collegamenti che contribuiscono maggiormente al termine di errore sono quelli che richiedono la regolazione maggiore. Tuttavia, i pesi non dovrebbero essere modificati troppo drasticamente, questo perché il termine di errore viene calcolato solo per l'esempio di addestramento corrente. In caso contrario, le modifiche apportate nelle iterazioni precedenti verrebbero annullate. Il **tasso di apprendimento**  $\lambda$ , il cui valore è compreso tra 0 e 1, può essere utilizzato per controllare la quantità di aggiustamenti effettuati in ciascuna iterazione:

- se  $\lambda$  è vicino a 0, il nuovo peso è perlopiù influenzato dal valore del vecchio peso;
- se  $\lambda$  è vicino a 1, il nuovo peso è sensibile alla quantità di regolazioni eseguite nell'iterazione corrente.

In alcuni casi, può essere utilizzato un  $\lambda$  adattivo: durante le prime iterazioni  $\lambda$  ha un valore grande, poi va progressivamente diminuendo.

È garantito che l'algoritmo di apprendimento perceptron converge verso una soluzione ottima per i problemi di classificazione separabili linearmente (purchè  $\lambda$  sia sufficientemente piccolo) [68]. In caso contrario l'algoritmo non è in grado di convergere (vedi Sezione 4.4.2).

#### 4.4.2 Multilayer Artificial Neural Network

Una rete neurale artificiale ha una struttura più complessa rispetto a quella di un modello percettrone semplice.

1. La rete, tra i livelli di input e di output, può contenere diversi livelli intermedi (vedi Figura 4.10). I livelli intermedi sono chiamati **hidden layer** o **livelli nascosti** e, i nodi incorporati in essi, sono chiamati **nodi nascosti**. La struttura risultante è nota come **rete neurale multistrato** (o multilivello). È possibile distinguere tra rete neurale **feed-forward** e rete neurale **ricorrente**.

In una rete neurale **feed-forward**, i nodi di un livello, sono collegati solo ai nodi del livello successivo.

In una rete neurale **ricorrente**, i collegamenti possono connettere sia nodi all'interno dello stesso livello, sia nodi ai livelli precedenti. Si noti che il perceptron

può essere visto come una rete neurale feed-forward a livello singolo (ha un solo livello di nodi, ovvero quello di output).

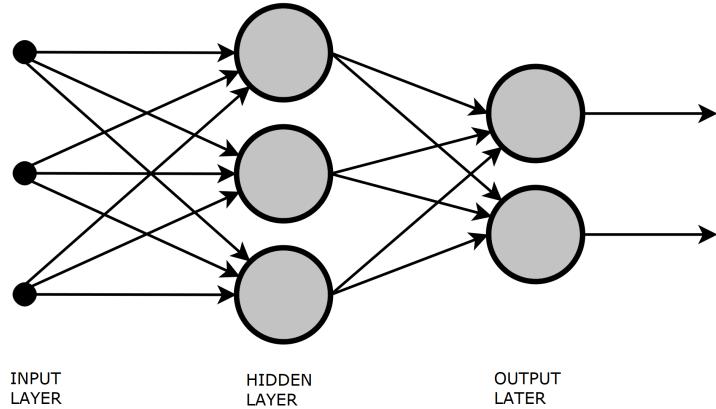


Figura 4.10: Esempio di una rete neurale artificiale multilayer feed-forward.

2. Una rete neurale può utilizzare diversi tipi di funzione di attivazione (a differenza del percepitrone che utilizza la sola funzione segno). Tra le principali funzioni di attivazione si hanno: funzioni lineari, iperboliche, tangenti e sigmoide. Queste funzioni di attivazione permettono, ai nodi nascosti e di output, di produrre valori di output non lineari rispetto ai parametri di input (la rete è in grado di modellare relazioni più complesse tra le variabili di input e di output).

**L’Apprendimento del Modello ANN.** La fase di apprendimento di una rete neurale artificiale differisce da quella di un singolo percepitrone. Basandosi su più livelli, determinare l’errore di previsione ( $y - \bar{y}$ ) associato a ciascun nodo risulta difficile. L’obiettivo dell’algoritmo di apprendimento, per una ANN, è quello di determinare un insieme di pesi che minimizza la somma totale degli errori al quadrato. La formula seguente (*funzione d’errore*) funge da guida durante la fase di apprendimento, ovvero durante la fase di aggiustamento dei pesi:

$$E(w) = \frac{1}{2} \sum_{i=1}^N (y_i - \bar{y}_i)^2. \quad (4.1)$$

Si noti che la Formula 4.1 dipende da  $w$  perchè la classe  $\bar{y}$  prevista è una funzione dei pesi assegnati ai nodi nascosti e di output. Algoritmi *golosi*, come quelli basati sul **metodo di discesa del gradiente**, sono stati sviluppati per risolvere in modo efficiente il problema dell'ottimizzazione [2, 73].

La **formula di aggiornamento del peso**, utilizzata dal metodo di discesa del gradiente, è così definita:

$$w_j \leftarrow w_j - \lambda \frac{\partial E(w)}{\partial w_j}, \quad (4.2)$$

dove  $\lambda$  è il *learning rate*. Il secondo termine della Formula 4.2, afferma che il peso dovrebbe essere incrementato in una direzione che riduce il termine di errore complessivo. Il gradiente è rappresentato da una serie di derivate parziali e, rappresenta il modo in cui la funzione cresce o decresce. Il metodo di discesa del gradiente può essere utilizzato per apprendere i pesi dei nodi di output e dei nodi nascosti. Tuttavia, calcolare il termine di errore  $\frac{\partial E}{\partial w_j}$ , per i nodi nascosti, senza sapere quali dovrebbero essere i valori di output, risulta difficile. Per risolvere questo problema è stata sviluppata una tecnica nota come **backpropagation**. In questa tecnica vi sono due fasi in cui si suddivide ogni iterazione:

- **fase in avanti:** in cui i pesi ottenuti dalla precedente iterazione vengono utilizzati per calcolare il valore di output di ogni neurone della rete (prima  $k$ , poi  $k + 1$ );
- **fase a ritroso:** la formula di aggiornamento del peso è applicata nella direzione opposta (prima  $k + 1$ , poi  $k$ ).

L'approccio di retropropagazione consente di utilizzare gli errori per i neuroni al livello  $k + 1$  per stimare gli errori al livello  $k$ .

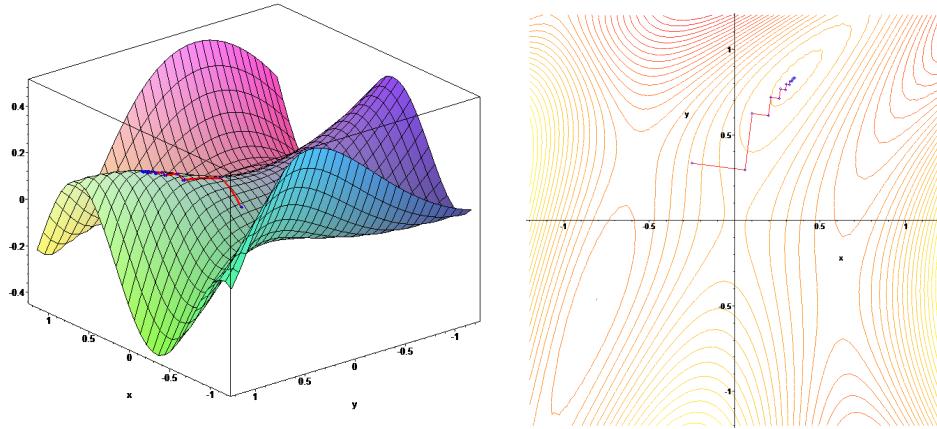


Figura 4.11: Esempio di superficie di errore con applicazione del metodo di discesa del gradiente [29] [30].

**Problemi di progettazione nell'apprendimento di una ANN.** Prima di procedere all'addestramento di una rete neurale, vanno presi in considerazione i seguenti problemi.

- Il numero dei nodi di input deve essere determinato assegnando un input per ogni variabile binaria o continua. Nel caso di variabile categorica, si deve avere un nodo per ogni valore o codificare la variabile *k-aria* in  $\log_2 k$  nodi di input.
- Il numero di nodi nel livello di output deve essere stabilito a seconda della tipologia del problema. Ad esempio, in un problema di classificazione a 2 classi è sufficiente un singolo nodo di output, mentre per problemi di *k*-classi ci sono *k* nodi di output.
- I pesi e il *bias* devono essere inizializzati e solitamente viene fatto assegnando valori casuali.
- Gli esempi di training con valori mancanti dovrebbero essere rimossi.

Allo stesso modo di altri modelli di Machine Learning, le ANN, possono soffrire di overfitting se la rete è troppo grande.

Le reti neurali artificiali sono in grado di gestire feature ridondanti, in quanto i pesi sono automaticamente appresi nella fase di allenamento e, nel caso di funzionalità ridondanti l'aggiornamento è irrilevante. Per lo stesso motivo, nel caso di dati di rumore

potrebbe esserci un eccessivo aggiornamento dei pesi, aumentando l'errore. Un'altra caratteristica a cui prestare attenzione è che il metodo di discesa del gradiente può convergere verso minimi locali.

La costruzione di un modello e il corrispettivo allenamento, soprattutto per problemi complessi, può risultare molto costoso. Tuttavia la fase di test successiva è estremamente rapida e veloce.

#### 4.4.3 Long Short-Term Memory

Per **Long Short-Term Memory** si intende un particolare modello di rete neurale ricorrente (RNN) [37]. Le classiche reti neurali ricorrenti soffrono di memoria a breve termine e, nel caso in cui si abbia una sequenza di informazioni molto longeva, hanno difficoltà a trasferire le informazioni dai passaggi temporali precedenti a quelli successivi. In particolare, nel caso di problemi basati su serie temporali, le classiche reti neurali ricorrenti non garantiscono dei buoni modelli predittivi. Questo perchè nella fasi di allenamento, utilizzando la backpropagation, il gradiente può erroneamente assumere valori che tendono a zero (*problema del gradiente evanescente* [67]) o, assumere valori enormi, a causa dei calcoli coinvolti nel processo (sono utilizzati valori a precisione finita, continui arrotondamenti possono aumentare di molto il termine di errore nei calcoli). Un valore per il gradiente troppo piccolo non permette ai pesi di adattarsi correttamente in fase di apprendimento, mentre un gradiente troppo grande rende il modello instabile (i pesi sono aggiornati a passi troppo grandi). Le reti neurali ricorrenti che utilizzano livelli LSTM risolvono parzialmente il problema, poichè quest'ultimi consentono al gradiente di fluire anche invariato durante la back-propagation.

Una **serie temporale**, in statistica descrittiva, si definisce come un insieme di variabili casuali ordinate secondo un ordine di tempo. Esprimono dunque, la dinamica di un certo fenomeno nel tempo. Le caratteristiche di reti LSTM permettono di studiare e interpretare al meglio problemi su serie temporali, permettendo di effettuare previsioni su fenomeni futuri, tenendo in considerazione elementi fondamentali come stagionalità e ciclicità.

Il problema della memoria a breve termine delle classiche RNN è risolto dalle LSTM

grazie al loro meccanismo di **gating**. In una cella di una RNN tradizionale, l'input a un dato passo temporale e lo stato nascosto del passo precedente, vengono processati solitamente attraverso una funzione di attivazione *tanh*, per ottenere il nuovo stato nascosto e l'output (vedi Figura 4.12).

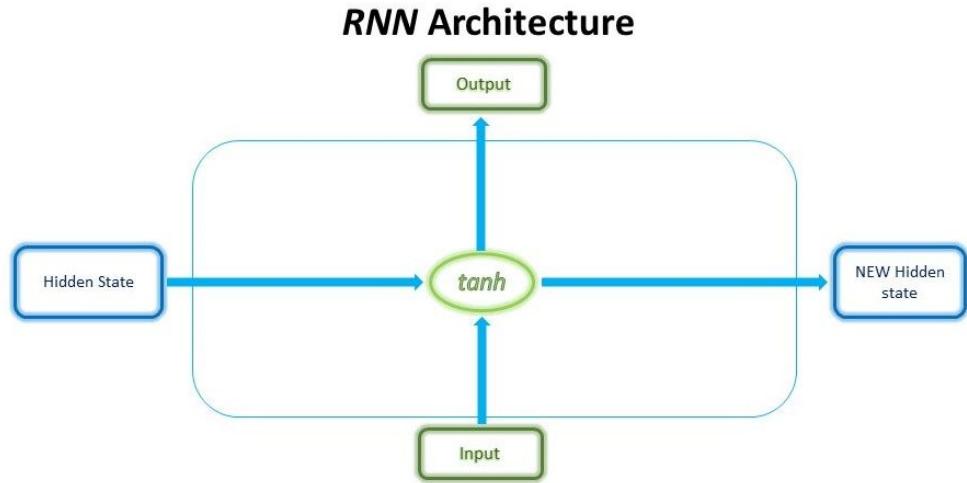


Figura 4.12: Funzionamento interno di una cella RNN [47].

In una LSTM, ad ogni passo, la cella utilizza 3 diversi tipi di informazione: *i dati di input correnti, la memoria a breve termine (dalla cella precedente) e la memoria a lungo termine* (vedi Figura 4.13). La memoria a breve termine rappresenta lo stato nascosto e la memoria a lungo termine è generalmente nota come stato della cella. Ogni cella utilizza le **porte** (gates) per regolare le informazioni, da conservare o scartare ad ogni passaggio, prima di passare le informazioni a lungo e a breve termine alla cella successiva. Le porte svolgono una funzione di *filtraggio*. Il ruolo di queste porte è di rimuovere selettivamente qualsiasi informazione irrilevante e far passare solo le informazioni utili ai fini del problema. Naturalmente, le porte necessitano di essere addestrate a filtrare accuratamente ciò che è utile e ciò che non lo è. Vi sono tre tipologie di porte, chiamate: **Input Gate**, **Forget Gate** e **Output Gate**.

## LSTM Architecture

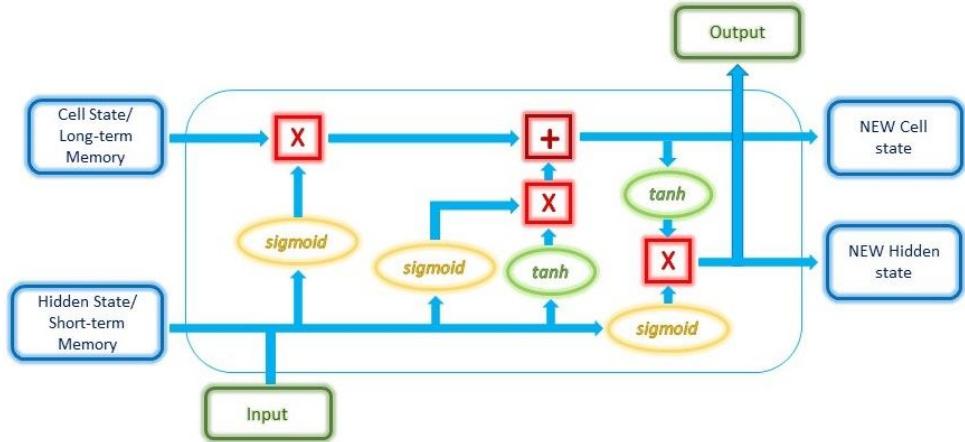


Figura 4.13: Funzionamento interno di una cella LSTM [47].

**Input Gate.** Decide quali nuove informazioni verranno memorizzate nella memoria a lungo termine. Funziona con i dati derivanti dall'input corrente e dalla memoria a breve termine (del passaggio temporale precedente) e, ne filtra le informazioni che non sono utili. Il processo è svolto da due livelli diversi:

1. Il primo livello funge da filtro che seleziona quali informazioni scartare e quali mantenere. Una funzione *sigmoide* processa la memoria breve termine e l'input corrente, restituendo un valore tra 0 e 1, con 0 indica che l'informazione non è importante, con 1 indica che l'informazione verrà utilizzata. In fase di allenamento, durante la backpropagation, i pesi nella funzione sigmoide vengono aggiornati in modo da perfezionare il lavoro di filtraggio.
2. Il secondo livello processa la memoria a breve termine e l'input corrente attraverso una funzione di attivazione (come la funzione *tanh*) per regolare la rete.

Gli output dei due livelli, vengono utilizzati per determinare l'output e preservati nella memoria a lungo termine (vedi Figura 4.14).

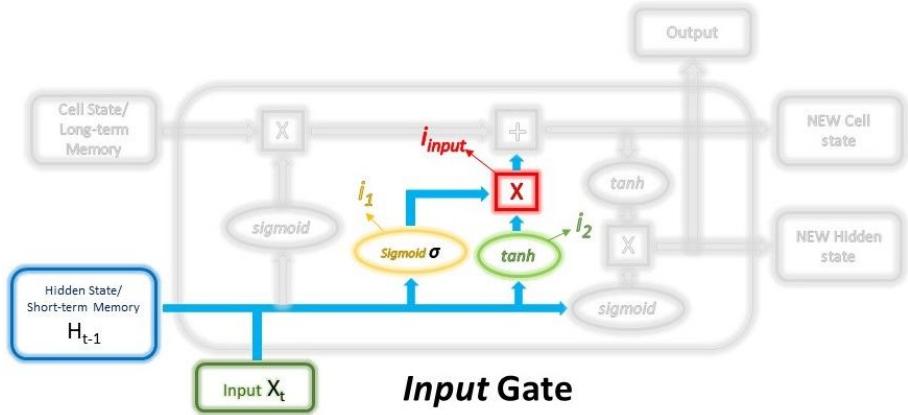


Figura 4.14: Rappresentazione del funzionamento dell'Input Gate [47].

**Forget Gate.** Decide quali informazioni, della memoria a lungo termine, devono essere conservate e quali scartate. Questo viene fatto moltiplicando la memoria a lungo termine in arrivo con un *vettore di dimenticanza*, generato dall'input corrente e dalla memoria a breve termine in arrivo attraverso una funzione sigmoide. Come nell'Input Gate, anche il vettore di dimenticanza funge da filtro selettivo. Il vettore di dimenticanza è composto dai valori 0 e 1 e viene successivamente moltiplicato con la memoria a lungo termine per scegliere quali informazioni di essa devono essere conservate. I valori forniti dai livelli Input e Forget sono combinati in una *nuova* versione della memoria a lungo termine, la quale è trasmessa alla cella successiva e utilizzata nell'Output Gate.

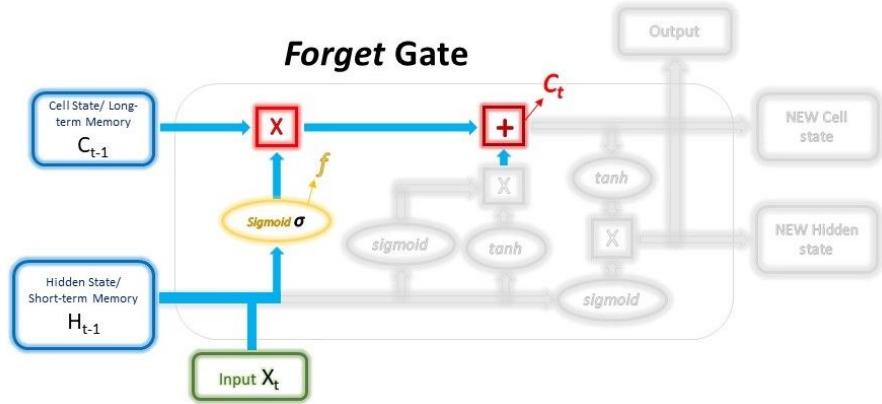


Figura 4.15: Rappresentazione del funzionamento del Forget Gate [47].

**Output Gate.** Utilizza l'input corrente, la precedente memoria a breve termine e la memoria a lungo termine (calcolata dal Forget Gate) per produrre il nuovo stato (memoria a breve termine) che sarà utilizzato dalla cella al passo temporale successivo e l'output della cella. Un ulteriore filtro utilizza una funzione signomide processando l'input corrente e la memoria a breve termine. L'output del filtro viene moltiplicato con la nuova memoria a lungo termine (elaborata in precedenza attraverso una funzione di attivazione  $tanh$ ), ottenendo la nuova memoria a breve termine. La memoria a breve e a lungo termine prodotta da queste porte viene trasferita alla cella successiva per la ripetizione del processo. L'output di ogni fase temporale può essere ottenuto dalla memoria a breve termine (nota anche come stato nascosto).

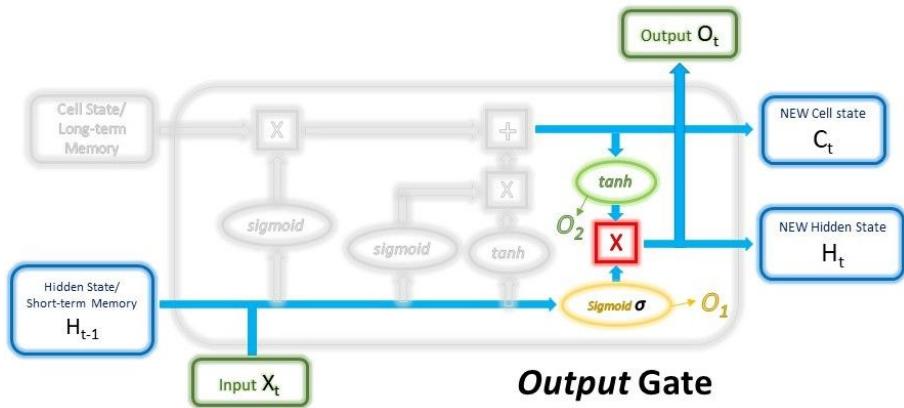


Figura 4.16: Rappresentazione del funzionamento dell'Output Gate [47].

È possibile implementare reti neurali ricorrenti con livelli LSTM utilizzando la libreria **TensorFlow** [1]. L'utente ha la possibilità di modificare una varietà di parametri, tra cui le funzioni di attivazione e il numero di unità per ogni livello. Tuttavia, per trovare la giusta ottimizzazione dei parametri, per il problema da affrontare, è necessario un lungo processo di test.

Nel Capitolo 6 saranno messe in confronto differenti implementazioni per previsioni di prezzo del Bitcoin utilizzando una rete LSTM e modelli SVM e Random Forest.

## 4.5 Overfitting

Con il termine overfitting, si intende un *sovradattamento* (*adattamento eccessivo*) di un modello di apprendimento automatico, nei confronti dei dati di allenamento. L'algoritmo individua delle false caratteristiche sui dati di training (ad esempio su dati di rumore) perdendo in *generalità*. Gli errori commessi durante la fase di previsione sono di due tipologie:

- **errori di training**, ovvero le discrepanze tra la previsione e il valore reale, in fase di allenamento del modello (nella classificazione il modello sbaglia a predire la classe  $y$  di un record  $x$ );
- **errori di generalizzazione**, per cui si intende l'errore atteso (previsto) sui record non ancora testati e mai visti prima dal modello.

L'overfitting è l'utilizzo di modelli che violano il *principio della parsimonia* o la legge di Occam Razor [33]. Secondo questo principio, i modelli apportano una complessità eccessiva rispetto a quella richiesta dal problema e, tra due modelli con lo stesso errore di generalizzazione, è sempre preferibile scegliere il più semplice.

Quando si verifica overfitting, le previsioni in fase di allenamento diventano estremamente accurate ma, quando il modello viene testato su dati nuovi, l'errore di generalizzazione è elevato. Nella costruzione di un buon modello, si ha l'obiettivo di minimizzare entrambi i tipi di errore. Un modello troppo complesso (vedi Figura 4.17) è molto più suscettibile a cadere in overfitting, rispetto ad un modello relativamente semplice. Questo perché riesce ad apprendere delle caratteristiche sui dati di allenamento che vanno oltre il concetto di generalizzazione. Allo stesso tempo, anche un modello semplice può cadere in overfitting se sottoposto ad un allenamento eccessivo o, se il dataset di training è molto piccolo.

È possibile diminuire sensibilmente il rischio di un modello di cadere in overfitting mediante varie metodologie.

- **Allenare il modello con più dati**: più dati di allenamento vengono forniti al modello e meno è probabile che si adattino troppo (la computazione richiesta per adattarsi eccessivamente a una grande mole di dati è maggiore).
- **Feature selection**: non tutti gli algoritmi sono in grado di selezionare in modo autonomo le funzionalità più rilevanti al fine della previsione, in questo caso

l’utente prima di implementare un modello deve predisporre con cura un buon dataset di allenamento.

- **Non allenare eccessivamente un modello**, in questo modo è possibile anticipare un adattamento eccessivo sui dati di allenamento. Utilizzando un set di valutazione, durante la fase di allenamento di una rete neurale, è possibile osservare le curve di apprendimento e, osservandone le metriche di valutazione (vedi Sezione 4.6), si può agire in anticipo sull’overfitting (vedi Figura 4.18).
- **Metodi ensemble**, i quali permettono ai modelli di concentrarsi singolarmente su sottoinsiemi di feature. In questo modo la complessità richiesta da ciascun singolo modello diminuisce (vedi Sezione 4.2).
- **Riduzione della complessità del modello**: utilizzando tecniche come il *Dropout* (nelle reti neurali) o il *pruning* (su alberi decisionali). Gli approcci di pruning sono solitamente due: pre-pruning o post-pruning. Il pre-pruning si limita a fermare la creazione dell’albero sotto determinate condizioni per evitare una eccessiva specializzazione (esempio massima dimensione dell’albero). Il post-pruning invece raffina un albero già creato, eliminando quei rami che non soddisfano alcune condizioni su un validation set precedentemente selezionato [50].

La situazione opposta all’overfitting, è detta **underfitting** (sottoadattamento). Può verificarsi nel caso in cui un modello deve ancora apprendere la struttura dei dati. Di conseguenza si comporta male sia nella fase di allenamento, sia nella fase di test. Nel caso in cui il modello implementato sia troppo semplice, per il problema da affrontare, si ha facilmente un sottoadattamento sui dati.

Nel Capitolo 6, sono mostrate varie tecniche utilizzate per prevenire overfitting e migliorare il tasso di generalizzazione dei modelli implementati.

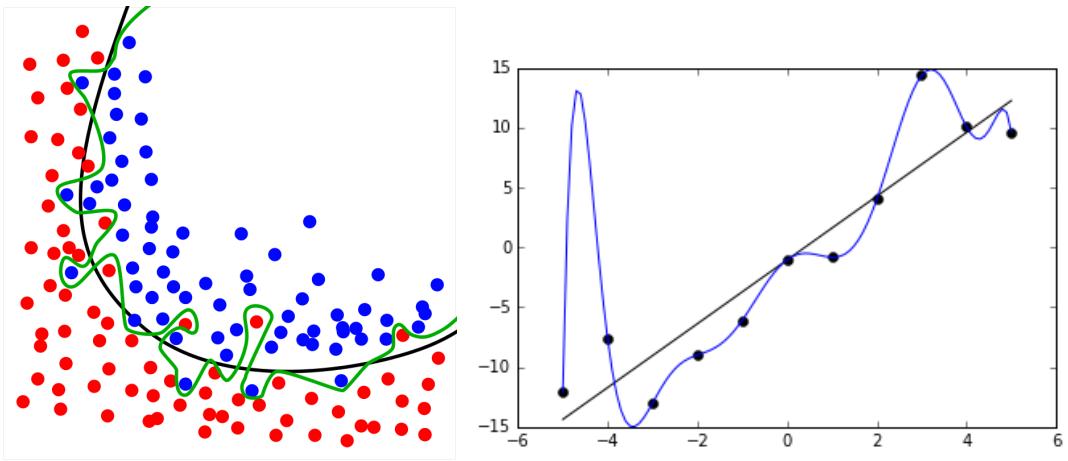


Figura 4.17: Nell’immagine di sinistra, la linea verde rappresenta un modello con overfitting, mentre la linea nera rappresenta un modello con un buon tasso di generalizzazione. Nell’immagine di destra, la linea blu rappresenta un modello che ha appreso anche caratteristiche sui dati di rumore, portandolo ad un sovraccarico, mentre la linea nera rappresenta invece un modello molto più semplice in grado di generalizzare meglio [15, 16].

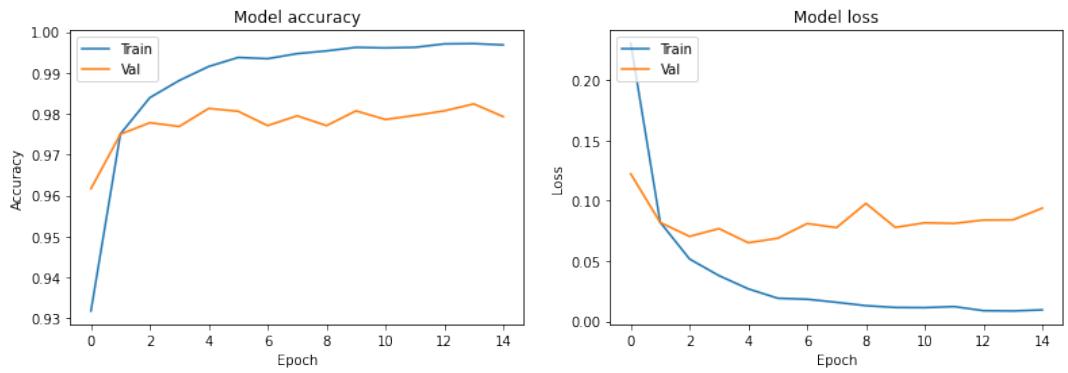


Figura 4.18: Esempio di curve di apprendimento di una rete neurale utilizzando 14 epoche. Nell’immagine a sinistra è rappresentata l’accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell’immagine a destra si ha l’errore di previsione.

## 4.6 Valutazione di un modello

La valutazione dei modelli di Machine Learning avviene attraverso delle metriche che permettono di misurare le performance. A seconda della tipologia del modello, *classificatore* o *regressore*, sono necessarie metriche diverse.

**Classificazione.** La valutazione delle performance relative ad un classificatore sono basate sul rapporto tra il numero di record di test correttamente e incorrettamente classificati. La *matrice di confusione* contiene e raccoglie tutte le informazioni sulle previsioni di un modello (previsioni corrette e errate su ciascuna classe). Nel caso di un compito di classificazione con  $n$  classi, la matrice di confusione è una matrice quadrata di dimensione  $n \times n$  (vedi Figura 4.19).

Confusion matrix for binary classification			
Actual value	A	TP	FN
	B	FP	TN
	A	B	
Predicted value			

Figura 4.19: Matrice di confusione per problema di classificazione binaria dove A e B sono rispettivamente le due classi [72].

Gli elementi chiave di una matrice di confusione sono:

- **True positive (TP):** numero di oggetti etichettati correttamente come appartenenti alla classe.
- **False positive (FP):** numero di oggetti etichettati erroneamente come appartenenti alla classe.
- **False negative (FN):** numero di oggetti che non sono stati etichettati come appartenenti alla classe ma dovrebbero esserlo.

- **True negative (TN)**: numero di oggetti etichettati correttamente come non appartenenti alla classe.

Grazie alla matrice di confusione è possibile determinare metriche, oltre ad accuratezza e rate di errore, che consentono di andare più in profondità nella valutazione di un modello:

- **Accuratezza**:  $\frac{\text{numero di previsioni corrette}}{\text{numero di previsioni totali}}$ .
- **Tasso di errore**:  $\frac{\text{numero di previsioni errate}}{\text{numero di previsioni totali}}$ .
- **Precision**:  $\frac{TP}{TP + FP}$ , misura la bontà di un modello quando la predizione è positiva.
- **Recall**:  $\frac{TP}{TP + FN}$ , misura la capacità di un modello nel prevedere correttamente le classi positive.
- **F1 Score**:  $2 \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$ , è la media ponderata di precision e recall.

**Regressione.** Le metriche utilizzate per problemi di classificazione, non sono utilizzabili nel momento in cui il modello è un regressore. Le previsioni effettuate da questa tipologia modello non sono valori categorici o discreti, ma valori continui, non è quindi possibile valutare le sue performance in termine di accuratezza, precision e recall. La valutazione nel caso di una regressione è basata sul determinare quanto il valore predetto dal modello si discosta dal valore reale. Le metriche principali sono:

- **R-Squared**: valuta la dispersione dei punti dati attorno alla linea di regressione adattata,

$$\frac{\text{Varianza del modello}}{\text{Varianza totale}}.$$

- **Mean Squared Error(MSE)/ Root Mean Squared Error(RMSE)**:

$$MSE(t) = \frac{N(1)}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2,$$

misura la media dei quadrati degli errori, ovvero la differenza quadratica media tra i valori stimati e il valore effettivo.  $RMSE$  è il valore ottenuto dalla radice quadrata di  $MSE$  e, rappresenta la distanza delle previsioni dai valori reali nel grafico di regressione (misurata mediante la distanza euclidea).

- **Mean Absolute Errore(MAE):**

$$MAE(t) = \frac{N(1)}{n} \sum_{i=1}^n |(\bar{y}_i - y_i)^2|,$$

invece della somma dei quadrati dell'errore in  $MSE$ ,  $MAE$  prende la somma del valore assoluto dell'errore. Misura la distanza media assoluta tra i dati reali e i dati previsti.

La valutazione dei modelli avviene sui dati di test. Il modello si approccia a questi record come farebbe nel caso pratico, con la differenza che in questo caso si ha in anticipo a disposizione il valore o la classe reale. È così possibile utilizzare le metriche di valutazione e, di conseguenza, ottimizzare gli iperparametri migliorando il modello.

# Capitolo 5

## Ricerche correlate

### 5.1 La teoria del mercato efficiente

L'obiettivo di ogni singolo trader o investitore è quello di riuscire a trarre profitto dalla compravendita di asset finanziari, sfruttando le varie fluttuazioni di mercato.

Una strategia **Buy and Hold**, si riferisce ad un approccio, nei mercati, in cui si compra un determinato asset e lo si detiene per un lungo periodo di tempo, ignorando movimenti e variazioni di prezzo intermedie. Solo alla fine del periodo di investimento, precedentemente fissato, si valuta la performance della strategia apportata. Un approccio di questo tipo si basa su un orizzonte temporale a lungo termine, il quale può essere prolungato anche per più di 10 anni.

Una strategia di **market timing**, invece, è una metodologia in cui il trader o investitore effettua numerose operazioni di acquisto e vendita, cercando di intercettare e sfruttare variazioni e trend di mercato. L'obiettivo è quello di agire sulle inefficienze di mercato, traendone profitto, battendo le performance di quest'ultimo. Un approccio di questo tipo tiene in considerazione le fluttuazioni soprattutto nel breve periodo, in quanto la gestione del portafoglio è molto attiva (sono eseguite molte operazioni). Nota anche con l'acronimo "EMH", la **teoria del mercato efficiente**, afferma che, poiché i mercati rispecchiano tutte le informazioni attualmente note, battere il mercato in maniera significativa è impossibile. In base a questa premessa, non ci sono asset sotto o sopravvalutati; tutto è prezzato esattamente come dovrebbe. Di conseguenza, non c'è modo o opportunità per generare dei ritorni extra. Sono pochi gli investitori che possono vantare la fama di essere riusciti a battere il mercato per un lungo perio-

do di tempo. Investitori come Warren Buffet e Peter Lynch - forte sostenitore degli investimenti *buy and hold* - hanno costruito la loro fortuna scegliendo i titoli giusti e battendo il mercato per molto tempo, il che dimostra che effettivamente è possibile [39]. Riuscire a determinare quei titoli in grado di aumentare il loro valore nell'arco di anni, a volte anche decenni, risulta essere molto difficile anche per gli investitori più esperti. Tra le altre problematiche per cui battere il mercato risulta essere impossibile, vi sono le commissioni di compravendita. Ogni volta che si effettua un'operazione nei mercati, che sia di acquisto o di vendita, si paga una commissione. Questo, nel caso di un numero elevato di operazioni, ha un costo notevole e porta ad abbassare eventuali performance di profitto di diversi punti percentuali. Con tutti questi livelli di difficoltà, che gli investitori devono superare, riuscire a battere il mercato è effettivamente molto difficile. Tuttavia, non sempre è possibile effettuare una strategia di *buy and hold*, in cui si alloca nel mercato una parte del proprio portafoglio prevedendo di non avere necessità, nel breve termine, di tale somma. In certi casi, potrebbe comunque essere preferita una gestione attiva dei propri fondi, in quanto si prevedono determinate spese imminenti. Con l'intento di contrastare fenomeni come l'inflazione, anche per periodi brevi come un singolo anno, effettuare una strategia di *buy and hold* potrebbe rivelarsi controproduttivo. Questo perchè fluttuazioni di mercato nel breve periodo potrebbero comunque riportare performance negative sul portafoglio, nonostante sul lungo termine la probabilità che l'intero mercato abbia performance positive è molto alta. In casi in cui l'obiettivo principale è la salvaguardia del portafoglio, anche nel breve termine, potrebbe rivelarsi utile una gestione attiva, che riesca soprattutto a garantire una performance non negativa. L'investitore, azienda, o singolo individuo ha quindi varie possibili strategie applicabili, migliori a seconda dei casi:

- **buy and hold**, nel caso in cui i fondi allocati molto probabilmente non si rivelano necessari per un utilizzo immediato;
- **gestione attiva**, in cui prevede che i fondi possano essere necessari.

Come verrà poi espresso a pieno nel Capitolo 7, l'ambito di studio di questa tesi è quello di fornire un sistema predittivo per una gestione attiva di portafogli allocati in criptovalute, in questo caso in Bitcoin, con la consapevolezza che battere il mercato, soprattutto nel lungo termine, sia estremamente difficile.

## 5.2 Ricerche correlate

Nel 2016, Rajashree Dash e Pradipta Kishore Dash [17] pubblicano uno studio sull'utilizzo del Machine Learning per implementare un modello di trading per il mercato azionario. Il modello implementato è costituito da una particolare rete neurale, chiamata *CEFLANN*, e utilizza indicatori tecnici finanziari per generare un segnale. Quest'ultimo viene passato ad un sistema decisionale di supporto che, analizzando il trend di mercato, mediante regole di trading fornisce in output una decisione (*buy*, *sell*, *hold*). Le prestazioni della rete neurale vengono paragonate con altri modelli di Machine Learning, come SVM e alberi decisionali. Come altra metrica di paragone, per determinare la bontà del modello, viene utilizzata una strategia di trading in cui si determina un segnale sulla base di singoli indicatori tecnici finanziari, come la media mobile a 15 periodi. I risultati ottenuti dimostrano che una rete neurale, che opera con multipli indicatori finanziari, performa meglio di modelli SVM, KNN e alberi decisionali. Tuttavia, ogni applicazione di Machine Learning implementata, permette di ottenere performance di profitto migliori rispetto ad una semplice strategia di trading basata su un singolo indicatore finanziario.

In quegli anni, l'interesse ad applicare modelli predittivi per l'andamento dei prezzi si è orientato non solo al mercato azionario, ma anche al mercato delle criptovalute. Alex Greaves e Benjamin Au, provano a predirre il prezzo di Bitcoin implementando reti neurali e SVM, paragonando le performance di quest'ultimi a tecniche di regressione lineare semplice [31]. Per allenare i modelli, sono stati utilizzati dati relativi al prezzo e alle informazioni presenti sulla blockchain di Bitcoin. Analizzando la catena di blocchi sono stati in grado di individuare importanti caratteristiche sulla rete, come: numero di utenti, grandi detentori di moneta e dati sulle transazioni. L'idea era di individuare una correlazione tra queste informazioni con il prezzo di mercato di bitcoin. I risultati di previsione migliori sono stati riscontrati con l'utilizzo di reti neurali, le quale hanno fornito un'accuratezza del 55,1%. Altri modelli come le macchine a vettori di supporto hanno presentato una precisione sulla previsione del 53,7% e un valore di MSE pari a 1,98 (su modello SVM di regressione). Paragonando i modelli ad apprendimento automatico con metodi di regressione semplici, i risultati dimostrano che i primi danno risultati migliori.

Nel 2018, metodi statistici di regressione lineare, come ARIMA, sono stati ulteriormente confrontati con algoritmi di Machine Learning. Sean McNally, Jason Roche,

Simon Caton hanno sviluppato modelli RNN e LSTM dimostrando nuovamente risultati migliori rispetto ad ARIMA, ottenendo un'accuracy di quasi il 53% nel caso migliore. I modelli sono stati allenati con feature relative al prezzo (prezzo di apertura, chiusura, massimo e minimo) e indicatori tecnici, come la media mobile semplice [49].

Lo stesso anno, Thearasak Phaladisailoed e Thanisa Numnonda [56] hanno implementato dei regressori utilizzando algoritmi di Machine Learning e reti neurali LSTM e GRU, ottenendo rispettivamente un MSE di 0.00002 e un  $R^2$  di 0.992 nel caso migliore. Il dataset utilizzato è composta da informazioni sul prezzo e sui volumi delle transazioni.

L'anno seguente, uno studio di Suhwan Ji, Jongmin Kim and Hyeonseung Im ha riscontrato che i modelli performano meglio nel caso di un problema di classificazione rispetto alla regressione [41]. Utilizzando un dataset costruito includendo anche informazioni relative alla blockchain, classificatori basati su reti neurali (DNN, LSTM, CNN) raggiungono un'accuracy di circa il 53% nella previsione di prezzo del bitcoin. Nel 2020, Mohammed Mudassir, Shada Bennbaia, Devrim Unal [52], ottengono buoni risultati implementando modelli SVM e reti neurali LSTM. Mudassir e i suoi colleghi, hanno incentrato lo studio nel prevedere i movimenti di prezzo anche nel breve periodo (dai 7 ai 90 giorni nel futuro). I risultati ottenuti evidenziano l'importanza di usare anche indicatori tecnici finanziari nell'ambito della previsione di prezzo delle criptovalute. I modelli implementati hanno ottenuto un'accuracy di circa il 65%.

La bassa accuracy raggiunta, ad oggi nelle ricerche, conferma ulteriormente la difficoltà di previsione e del problema stesso, in un mercato altamente volatile e molto influenzabile da fattori esterni non prevedibili. Un esempio eclatante è quanto avvenuto a febbraio 2021, in cui Tesla annunciò di detenere e accettare bitcoin come forma di pagamento per le proprie auto [69]. Ciò, portò in pochi giorni il valore della criptovaluta ad aumentare notevolmente. Qualche mese dopo, Elon Musk, il CEO di Tesla, con un nuovo annuncio dichiara che la compagnia non accetterà più bitcoin a causa del negativo impatto ambientale del suo ecosistema. Tra tanti altri fattori, questo causò l'inizio di una decrescita di oltre il 40% del prezzo della criptovaluta in pochi giorni. Due eventi simili difficilmente sarebbero potuti essere previsti da un modello di Machine Learning, soprattutto nel primo caso. Nel secondo, data una

situazione di iperestensione del prezzo e di una crescita che si stava progressivamente appiattendo, un modello avrebbe potuto individuare un possibile calo futuro del prezzo (difficilmente sarebbe stato in grado di prevedere un dump del 40%). Nonostante tutto questo, varie ricerche, hanno comunque dimostrato che risulta possibile ottenere un profitto, sulla compravendita di bitcoin, grazie sistemi implementati con l'utilizzo dell'apprendimento automatico. Nella Tabella 5.1 vi è un breve riepilogo dei risultati ottenuti ad oggi da altri ricercatori e studiosi nell'ambito della previsione dell'andamento di mercato del Bitcoin.

Ricerca	Risultati ottenuti
<b>Rajashree Dash e Pradipta Kishore Dash [17].</b>	Risulta possibile sovraperformare tecniche di trading, basate su analisi tecnica, utilizzando modelli di apprendimento automatico.
<b>Alex Greaves e Benjamin Au [31].</b>	I risultati ottenuti utilizzando reti neurali raggiungono un'accuracy di circa il 55,1%. Tecniche di Machine Learning sovraperfornano i regressori semplici.
<b>Sean McNally, Jason Roche, Simon Caton [49].</b>	Utilizzando feature relative al prezzo e ad altri indicatori tecnici finanziari è stata raggiunta un'accuracy di circa il 53%, mediante LSTM e RNN.
<b>Thearasak Phaladisailoed e Thansisa Numnonda [56].</b>	Implementando regressori basati su LSTM e GRU hanno raggiunto rispettivamente un MSE di 0.00002 e un $R^2$ di 0.992 nel caso migliore.
<b>Suhwan Ji, Jongmin Kim and Hyeonseung Im [41].</b>	Le performance dei classificatori risultano essere migliori rispetto ai regressori. Mediante classificatori basati su reti neurali (DNN, LSTM e CNN) raggiungono un'accuracy di circa il 53%.
<b>Mohammed Mudassir, Shada Bennbaia, Devrim Unal [52].</b>	Risulta possibile amplificare lo spettro di previsione dei modelli oltre il giorno successivo. Raggiunta accuracy di circa il 65%.

Tabella 5.1: Tabella di riepilogo ricerche correlate.

# Capitolo 6

## Ricerca ed Analisi

In questo capitolo vengono presentati e analizzati i risultati ottenuti nella ricerca di questa tesi. L'obiettivo è quello di implementare e dare una base, a ricerche e lavori futuri, nello sviluppo di modelli predittivi efficienti nell'ambito dei mercati delle criptovalute, in particolare Bitcoin.

Dopo un prima parte, in cui verranno presentate le tecniche di simulazione per testare la validità e le performance, i dati utilizzati e il loro pre-processamento, saranno descritti i modelli ideati. In totale, sono stati sviluppati e costruiti 3 tipi di modelli (regressori e classificatori), utilizzando: Random Forests, Macchine a Vettori di Supporto (SVM) e reti neurali di tipo Long Short-Term (LSTM).

### 6.1 Raccolta dati

Per poter effettuare un'analisi del mercato nel migliore dei modi, è necessario considerare sia l'analisi tecnica, che l'analisi fondamentale. Il datasat, utilizzato per allenare i modelli, necessita quindi di essere composto da metriche di entrambe le tipologie.

Gli **indicatori tecnici** utilizzati, descritti nel Capitolo 3, sono  $SMA_{50}$ ,  $SMA_{200}$ ,  $EMA_{12}$ ,  $EMA_{26}$ ,  $MACD$ ,  $Stochastic\ KD$ ,  $RSI$ ,  $WR_{14}$ .

I dati per l'**analisi fondamentale**, provengono da Glassnode [24], un provider di dati derivati mediante l'**analisi on-chain**. Studiando la blockchain di Bitcoin, è possibile trovare e definire molte metriche, le quali possono essere utilizzate per individuare correlazioni e pattern con il prezzo di mercato della criptovaluta. Come detto nel Capitolo 3, nell'analizzare le dinamiche di mercato delle criptovalute, le sole metri-

che di valutazione convenzionali (indicatori tecnici finanziari e i soli dati sul prezzo come apertura, chiusura) non sono sufficienti. La blockchain, genera costantemente un grande quantità di dati, accessibili liberamente e incorruttibili, i quali consentono di derivare misure precise e affidabili sull'attività economica della rete stessa. La raccolta dati, e la relativa generazione di metriche opportune, ha elevati costi operativi, in aggiunta alla necessità di avere un'infrastruttura completa di database e di esperti data scientists. Glassnode offre un catalogo di metriche accessibili liberamente, o a pagamento, su una vasta gamma di blockchain relative a differenti criptovalute. In questo caso specifico le metriche, per implementare il dataset, sono state raccolte mediante l'utilizzo di API, fornite dal provider stesso con un abbonamento *advanced* [25]. Ovviamente, non tutte le metriche derivabili possiedono una correlazione, diretta o inversa, con il prezzo di bitcoin. Come mostrato nel Sezione 6.2, dopo una prima fase di raccolta dati, è stato effettuato un processo di *feature selection* per restringere il dataset, mantenendo solo le più significative (vedi Sezione 6.2.1).

### 6.1.1 Descrizione Feature

Una breve descrizione di ogni metrica, di analisi fondamentale, recuperata è qui di seguito riportata. Tutti i dati, eccetto *google trends index*, provengono da Glassnode. L'indice di trend, in particolare, è stato ricavato mediante API direttamente da Google [32] (i parametri sono stati impostati su una copertura globale e per una ricerca su qualsiasi categoria).

<b>FEATURE</b>	<b>DESCRIZIONE</b>
<b>open</b>	Il valore di prezzo ad inizio giornata.
<b>high</b>	Il valore di prezzo più alto raggiunto durante la giornata.
<b>low</b>	Il valore di prezzo più basso raggiunto durante la giornata.
<b>close</b>	Il valore di prezzo a fine giornata.
<b>volume</b>	La quantità totale di monete trasferite sulla blockchain. Vengono conteggiati solo i trasferimenti andati a buon fine.
<b>new addresses</b>	Il numero di indirizzi univoci che sono apparsi per la prima volta nella rete in una transazione.

Tabella 6.1: Descrizione Feature parte 1.

FEATURE	DESCRIZIONE
<b>active addresses</b>	Il numero di indirizzi univoci attivi nella rete come mittente o destinatario. Vengono conteggiati solo gli indirizzi corrispondenti a transazioni confermate.
<b>total unique addresses</b>	Il numero totale di indirizzi univoci che sono mai apparsi nella blockchain in una transazione.
<b>sending addresses</b>	Il numero di indirizzi univoci attivi come mittente di fondi. Vengono conteggiati solo gli indirizzi corrispondenti a trasferimenti riusciti con ammontare diverso da zero.
<b>receiving addresses</b>	Il numero di indirizzi univoci attivi come destinatario di fondi. Vengono conteggiati solo gli indirizzi corrispondenti a trasferimenti riusciti con ammontare diverso da zero.
<b>total fees</b>	L'importo totale delle commissioni pagate ai miners. Le monete emesse (minate) non sono incluse.
<b>mean transaction fees</b>	La commissione media per transazione. Le monete emesse (minate) non sono incluse.
<b>median transaction fees</b>	La commissione mediana per transazione. Le monete emesse (minate) non sono incluse.
<b>number of transaction</b>	Il numero totale di transazioni effettuate. Vengono conteggiate solo le transazioni andate a buon fine.
<b>transaction rate</b>	Il numero di transazioni effettuate al secondo. Vengono conteggiate solo le transazioni andate a buon fine.
<b>median transfer volume</b>	La mediana di bitcoin scambiati ad ogni transazione. Vengono conteggiati solo i trasferimenti andati a buon fine.
<b>mean transfer volume</b>	La media di bitcoin scambiati ad ogni transazione. Vengono conteggiati solo i trasferimenti andati a buon fine.
<b>total transfer volume</b>	La quantità totale di monete trasferite sulla blockchain. Vengono conteggiati solo i trasferimenti andati a buon fine.
<b>total transaction size</b>	La dimensione totale (in bytes) di tutte le transazioni nell'intervallo di tempo (in questo caso è un intervallo giornaliero).

Tabella 6.2: Descrizione Feature parte 2.

FEATURE	DESCRIZIONE
<b>mean transaction size</b>	La dimensione media (in bytes) di tutte le transazioni nell'intervallo di tempo (in questo caso è un intervallo giornaliero).
<b>hash rate</b>	Il valore medio stimato di hash per secondo prodotto dalla rete di miners.
<b>mining difficulty</b>	L'attuale stima di hash necessari per estrarre un blocco. Nota: <i>la difficoltà di mining in Bitcoin è spesso indicata come la relativa difficoltà rispetto al blocco di genesi, che richiedeva circa <math>2^{32}</math> hash.</i>
<b>reserve risk</b>	Definito come il rapporto tra prezzo e monete holdate da un lungo periodo. Viene utilizzato per valutare la fiducia dei detentori a lungo termine rispetto al prezzo di bitcoin in un dato momento. Quando la fiducia è alta e il prezzo è basso, c'è un rapporto rischio/rendimento interessante per investire. Quando la fiducia è bassa e il prezzo è alto, il rischio/rendimento non è attraente.
<b>market cap</b>	La capitalizzazione di mercato, definita come il prodotto dell'offerta corrente per il prezzo corrente in USD.
<b>stock to flow</b>	Modello popolare che presuppone che la scarsità guida il valore. È definito come il rapporto tra lo stock attuale di una merce (offerta circolante di Bitcoin) e il flusso di nuova produzione (cioè i bitcoin appena estratti). Il prezzo di Bitcoin ha storicamente seguito il rapporto $S/F$ e quindi è un modello che può essere utilizzato per prevedere le future valutazioni.
<b>stock to flow deflection</b>	Rapporto tra prezzo corrente di Bitcoin e il modello $S/F$ . Se la deflessione è $\geq 1$ significa che Bitcoin è sopravvalutato secondo il modello $S/F$ , altrimenti sottovalutato.
<b>realized profit</b>	Indica la somma (valore in USD) di tutte le monete spostate, il cui prezzo nell'ultimo trasferimento era superiore al prezzo al movimento corrente.

Tabella 6.3: Descrizione Feature parte 3.

FEATURE	DESCRIZIONE
<b>realized loss</b>	Indica la somma (valore in USD) di tutte le monete spostate, il cui prezzo nell'ultimo trasferimento era inferiore al prezzo al movimento corrente.
<b>cvdd</b>	Quando le monete si spostano da un utente all'altro, nella transazione oltre al valore di moneta inviata, vi è un valore temporale relativo a quanto tempo l'utente originale ha detenuto le sue monete. Il CVDD tiene traccia della somma cumulativa di questi termini mentre le monete passano tra gli utenti in rapporto all'età del mercato. Storicamente, CVDD è stato un indicatore accurato per i minimi del mercato globale di Bitcoin [18].
<b>addresses&gt;001</b>	Il numero di indirizzi univoci che detengono almeno 0,01 btc.
<b>addresses&gt;01</b>	Il numero di indirizzi univoci che detengono almeno 0,1 btc.
<b>addresses&gt;1</b>	Il numero di indirizzi univoci che detengono almeno 1 btc.
<b>addresses&gt;10</b>	Il numero di indirizzi univoci che detengono almeno 10 btc.
<b>addresses&gt;100</b>	Il numero di indirizzi univoci che detengono almeno 100 btc.
<b>addresses&gt;1000</b>	Il numero di indirizzi univoci che detengono almeno 1000 btc.
<b>addresses&gt;10k</b>	Il numero di indirizzi univoci che detengono almeno 10000 btc.
<b>exchanges withdrawls</b>	Il totale di bitcoin prelevati dagli Exchanges. Se un grande ammontare di moneta viene prelevato dai wallet degli Exchanges è altamente probabile che sia per detenere i btc in wallet custodial (privati). Questo significa che tali bitcoin non possono essere venduti in modo diretto, portando il valore di offerta reale di moneta a scendere. Con una bassa offerta, un forte pressione di acquisto porterebbe il prezzo a salire velocemente.

Tabella 6.4: Descrizione Feature parte 4.

FEATURE	DESCRIZIONE
<b>exchanges deposits</b>	Il totale di bitcoin depositati sugli Exchanges. Se un grande ammontare di moneta viene depositato dai wallet privati agli Exchanges è altamente probabile che sia per vendere i btc. Questo significa che tali bitcoin possono essere venduti in modo diretto, portando il valore di offerta reale di moneta ad aumentare. Una forte offerta potrebbe causare un abbassarsi del prezzo.
<b>exchange outflow</b>	Il totale di transazioni che hanno come mittente indirizzi corrispondenti ad Exchanges.
<b>exchange inflow</b>	Il totale di transazioni che hanno come destinatario indirizzi corrispondenti ad Exchanges.
<b>mvr ratio</b>	Il valore di mercato rispetto al valore realizzato (MVRV) è il rapporto tra capitalizzazione di mercato e capitalizzazione realizzata. Fornisce un'indicazione di quando il prezzo negoziato è inferiore a un "valore equo".
<b>nvt signal</b>	Spesso chiamato "rapporto crittografico <i>PE</i> (price to earnings)". Sebbene le criptovalute non abbiano guadagni, si può sostenere che il valore totale delle transazioni, che fluiscono attraverso la rete, sia un indicatore di quanta utilità gli utenti derivano dalla blockchain. La metrica utilizzata in questa tesi è una versione modificata del rapporto NVT originale. Utilizza una media mobile a 90 giorni per il volume delle transazioni giornaliere al denominatore (invece del volume delle transazioni giornaliere tradizionale). La media mobile migliora l'indicatore, rendendolo di fatto un buon anticipatore.
<b>google trends index</b>	Indica la frequenza di ricerca sul web della parola Bitcoin.

Tabella 6.5: Descrizione Feature parte 5.

## 6.2 Realizzazione Dataset di allenamento

In totale, si hanno a disposizione 51 metriche. Ciò significa che, una volta aggregate le informazioni in un unico dataset, ogni record è costituito da 51 attributi. Il dataset, ordinato in ordine cronologico, contiene record a partire dal giorno 21/07/2013 fino al giorno 27/06/2021 (2896 record).

Tuttavia, una grande mole di dati a disposizione, nel Machine Learning, non sempre è un ottimo indicatore. Modelli di apprendimento automatico come Random Forests e SVM, possono risentire negativamente di un numero elevato di feature, qualora alcune di esse non aggiungano informazioni rispetto ad altre, oppure non siano correlate rispetto alla variabile da prevedere. Troppe caratteristiche possono influire negativamente sulle prestazioni del modello, sia in termini di tempo per l'addestramento, sia in termini di rischio di overfitting. Risulta quindi necessario individuare e selezionare quegli attributi che siano veramente in grado di fornire utilità, in materia di previsione, ai modelli.

### 6.2.1 Feature Selection

La **feature selection** (*selezione delle caratteristiche*, in italiano), è una tecnica utilizzata per selezionare un sottoinsieme di attributi rilevanti, per l'addestramento di un modello di apprendimento automatico. Tale selezione viene effettuata per diversi motivi:

- semplificazione dei modelli per renderli più facilmente interpretabili da ricercatori o utenti [40];
- tempi di training minori [46];
- miglioramento della generalizzazione e riduzione del rischio di incorrere in overfitting.

In certi casi, è possibile che una o più feature forniscano le stesse informazioni di altre, e nonostante vi sia una forte correlazione con la variabile da prevedere, è possibile rimuoverle senza causare una perdita di prestazioni.

Le principali tecniche di feature selection si basano su:

- **Metodi Wrapper:** sono algoritmi che mirano a trovare la migliore combinazione possibile di funzionalità, sulla base delle prestazioni del modello. Utilizzano un modello di apprendimento automatico, il quale viene addestrato con diversi possibili sottoinsiemi di feature, e successivamente testato su un set di controllo. In base alle performance, viene assegnato un punteggio ad ogni sottoinsieme. Poichè i metodi wrapper addestrano un nuovo modello per ogni sottoinsieme,

sono molto costosi computazionalmente, tuttavia forniscono il set di feature che permette di ottenere le migliori prestazioni per quel particolare tipo di problema. Un tipico esempio di metodo wrapper è rappresentato da Boruta (vedi Sezione 6.2.2).

- **Metodi Filtro:** mediante una misura statistica (ad esempio la correlazione), assegnano un punteggio ad ogni feature. La scelta degli attributi da utilizzare verterà sul valore del punteggio assegnato ad ognuna. Questi metodi sono solitamente poco costosi computazionalmente, ma producono un set di funzionalità che non è specifico per il tipo di modello. Tale mancanza di ottimizzazione, significa che un set di feature, ottenuto mediante un metodo di filtraggio, è più generale di un set derivato da un metodo wrapper, ed in genere forniscono prestazioni peggiori. I metodi filtro sono spesso utilizzati come fase di pre-elaborazione per i metodi wrapper. Sono metodi semplici ed efficaci, sono generalmente il primo tentativo utilizzato per ridurre la dimensione dei dataset. Alcuni metodi filtro, tra i più utilizzati, sono correlazione di Pearson e correlazione di Spearman (vedi Sezione 6.2.3).
- **Metodi Emnedded:** chiamati anche metodi incorporati. L'algoritmo di selezione delle caratteristiche è integrato come parte dell'algoritmo di apprendimento. La tecniche embedded più comuni sono gli algoritmi Decision Tree e Random Forests. Questi algoritmi selezionano una caratteristica in ogni fase ricorsiva del processo di crescita dell'albero e, dividono l'insieme campione in sottoinsiemi sempre più piccoli. Più nodi figli in un sottoinsieme sono nella stessa classe, più informative sono le funzionalità. L'algoritmo prova tutte le possibili modalità di suddivisione per le funzionalità e sceglie quella che suddivide meglio i dati. Lavora similmente al metodo wrapper, poiché vengono provate tutte le possibili combinazioni e viene scelta la migliore. Per la classificazione la scissione avviene in base al grado d'impurità, e per la regressione viene utilizzata la varianza (vedi Sezione 4.2.1). Altri metodi incorporati sono il *LASSO* con la penalità *L1* e *Ridge* con la penalità *L2* per la costruzione di un modello lineare. I metodi embedded prendono in considerazione l'interazione delle funzionalità e permettono una buona valutazione come i metodi wrapper, ma sono anche piuttosto veloci come i metodi filtro.

Per ridurre la dimensionalità del dataset sono stati utilizzati, in combinazione, metodi wrapper e metodi filtro. I risultati prodotti da ogni metodologia sono stati analizzati e valutati, con un sistema pesato, e le migliori feature sono andate a comporre il dataset di allenamento. La scelta di utilizzare un metodo wrapper si basa sul fatto che il semplice metodo filtro permette di individuare solamente eventuali caratteristiche ridondanti. Con quest'ultimo non è possibile determinare l'importanza effettiva di ogni feature nei confronti della previsione. Un metodo wrapper, invece permette di ottenere un valore di importanza per ogni funzionalità per lo specifico problema.

### 6.2.2 Boruta

In un metodo wrapper, un modello di apprendimento automatico viene utilizzato per ricavare una valutazione sulle caratteristiche [44]. È possibile utilizzare qualsiasi modello in grado di restituire l'importanza di ogni feature. Per un'ottimizzazione temporale e per ridurre le responsabilità dell'utente, il modello dovrebbe essere sia computazionalmente efficiente che semplice, e possibilmente senza parametri da impostare.

**Boruta** è un metodo di feature selection che utilizza un approccio wrapper costruito attorno a un modello di foresta casuale. L'algoritmo è un'estensione dell'idea introdotta da Stoppiglia, Dreyfus, Dubois e Oussar [63], in cui per determinare la rilevanza delle feature si confronta il dataset originario con dei sottoinsiemi di funzionalità.

L'algoritmo delle foreste è relativamente veloce, può essere eseguito in genere senza l'ottimizzazione dei parametri e fornisce una stima numerica dell'importanza delle caratteristiche. La misura dell'importanza di un attributo si ottiene considerando la perdita di accuratezza sulla classificazione, la quale viene calcolata separatamente per tutti gli alberi della foresta che operano con diversi sottoinsiemi di funzionalità.

**Shadow feature.** In Boruta, le caratteristiche non competono tra loro, ma con una loro versione *randomizzata*. Partendo dal dataset iniziale, viene creato un nuovo dataset mescolando casualmente, tramite permutazione, ogni feature. Il set di dati ottenuto, chiamato *dataset ombra*, viene successivamente unito al dataset originale. Tale unione dà luogo a un dataset con il doppio delle colonne dell'originale. Il nuovo dataset viene utilizzato per allenare un modello random forest e l'importanza di ogni feature viene confrontata ad una **soglia**. Quest'ultima è definita come l'importanza

più alta tra le shadow feature. Quando l'importanza di una delle caratteristiche del dataset originale è superiore alla soglia, gli viene assegnato un punto. L'idea è che una funzionalità è utile solo se è in grado di fare meglio della migliore funzionalità randomizzata. Quindi l'insieme di importanza degli attributi ombra viene utilizzato come riferimento per decidere quali attributi sono veramente importanti.

**Distribuzione Binomiale.** Per eliminare il fattore di casualità, il processo viene iterato per più volte. Dopo un numero  $n$  di iterazioni, il sistema valuta i punteggi ottenuti dalle funzionalità e seleziona le migliori. In Boruta, non c'è una soglia prestabilita per il punteggio secondo cui si stabilisce le feature da prendere o meno. La regola di selezione viene derivata dalla distribuzione binomiale delle prove effettuate (vedi Figura 6.1). Le funzionalità sono catalogate, in base al punteggio, in:

- **area di rifiuto:** le caratteristiche in essa sono considerate irrilevanti, vengono quindi eliminate;
- **area di indecisione:** boruta è indeciso sulle caratteristiche che si trovano in questa area, la decisione spetta all'utente;
- **area di accettazione:** le caratteristiche in essa sono considerate predittive, vengono mantenute.

Le aree sono definite selezionando le due porzioni più estreme della distribuzione chiamate code della distribuzione.

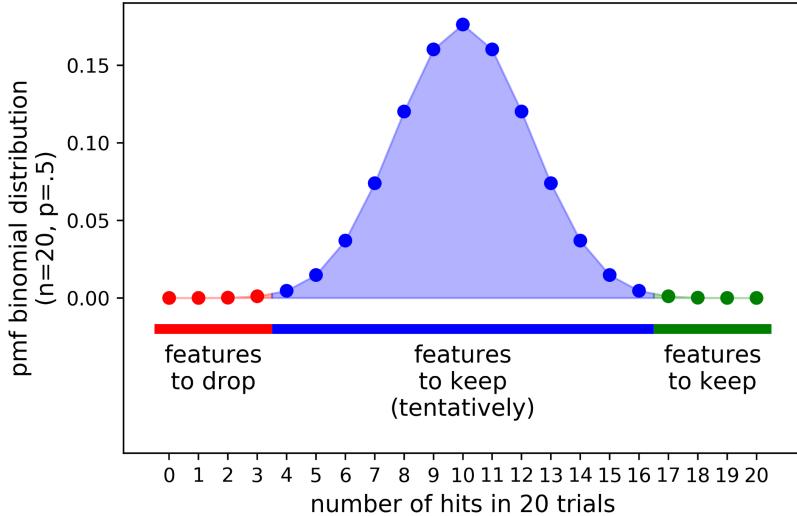


Figura 6.1: Distribuzione binomiale e posizionamento delle feature [48].

L’algoritmo di Boruta consiste quindi nei seguenti passaggi:

1. Estendere il dataset aggiungendo copie casuali di tutte le variabili (viene sempre esteso di almeno 5 attributi ombra, anche se il numero di attributi nell’insieme originale è inferiore a 5).
2. Allenare un modello basato su foresta casuale e determinare la soglia della miglior shadow feature.
3. Raccogliere i punteggi delle feature originali in base alla soglia.
4. Reiterare il processo più volte (a seconda del numero di feature e dei risultati ottenuti), in modo da prendere decisioni rafforzate dai test.
5. Selezionare ed eliminare le feature in base alla distribuzione binomiale.

Boruta potrebbe richiedere molto tempo per grandi set di dati, tuttavia, questo sforzo è essenziale per produrre una selezione statisticamente significativa di caratteristiche rilevanti.

### 6.2.3 Correlazione di Pearson e Spearman

I metodi basati su filtro permettono di individuare le feature ridondanti e di rumore (sono metodi generali, non si specificano su un particolare problema). Metodi di cor-

relazione come Pearson e Spearman, determinano un valore per ogni possibile coppia di caratteristiche. Il valore rappresenta la forza della correlazione tra le due feature.

**Correlazione di Pearson.** Per due variabili, restituisce un valore che indica l'intensità della correlazione. Ha un valore compreso tra  $+1$  e  $-1$ , dove  $+1$  corrisponde alla perfetta correlazione lineare positiva,  $0$  corrisponde ad un'assenza di correlazione lineare e  $-1$  corrisponde alla perfetta correlazione lineare negativa [54]. Date due feature  $X$  e  $Y$ , l'indice di correlazione di Pearson è definito come la loro covarianza divisa per il prodotto delle deviazioni standard:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y};$$

dove  $\sigma_{XY}$  è la covarianza tra  $X$  e  $Y$  e  $\sigma_X$   $\sigma_Y$  sono le due deviazioni standard.

- Se  $\rho_{XY} > 0$ , le variabili  $X$  e  $Y$  si dicono *direttamente correlate*, oppure *correlate positivamente*;
- se  $\rho_{XY} = 0$ , le variabili  $X$  e  $Y$  si dicono *incorrrelate*;
- se  $\rho_{XY} < 0$ , le variabili  $X$  e  $Y$  si dicono *inversamente correlate*, oppure *correlate negativamente* [70];

Inoltre valgono le seguenti affermazioni:

- se  $0 < |\rho_{XY}| < 0.3$  si ha correlazione **debole**;
- se  $0.3 < |\rho_{XY}| < 0.7$  si ha correlazione **moderata**;
- se  $|\rho_{XY}| > 0.7$  si ha correlazione **forte**.

Se le due variabili sono indipendenti allora l'indice di correlazione vale  $0$ . Non vale la conclusione opposta: in altri termini, l'incorrelazione è condizione necessaria ma non sufficiente per l'indipendenza [70].

**Correlazione di Spearman.** Il coefficiente di Spearman è una misura non parametrica della dipendenza statistica tra due variabili. Il coefficiente di Spearman esprime il livello di correlazione progressiva costante di due variabili. Diversamente

dal coefficiente di correlazione lineare di Pearson, il coefficiente di Spearman non misura una relazione lineare. Esso permette di stabilire quanto bene una relazione tra due variabili può essere descritta usando una funzione monotona. A livello pratico il coefficiente  $\rho$  è semplicemente un caso particolare del coefficiente di correlazione di Pearson, dove i valori vengono convertiti in ranghi prima di calcolare il coefficiente [71]. I valori ottenuti possono essere valutati allo stesso modo della correlazione di Pearson appena descritta.

Sia la correlazione di Pearson che la correlazione di Spearman forniscono come risultato un valore per ogni possibile coppia di feature, a differenza del metodo Boruta che ritorna un insieme di punteggi che rappresentano l'importanza di previsione nei confronti di una singola metrica  $y$ . L'insieme di valori ottenuto mediante l'utilizzo di Pearson e Spearman può essere rappresentato in una matrice quadrata simmetrica, detta **matrice di correlazione**, in cui gli indici di riga e colonna sono le feature (vedi Figura 6.2).

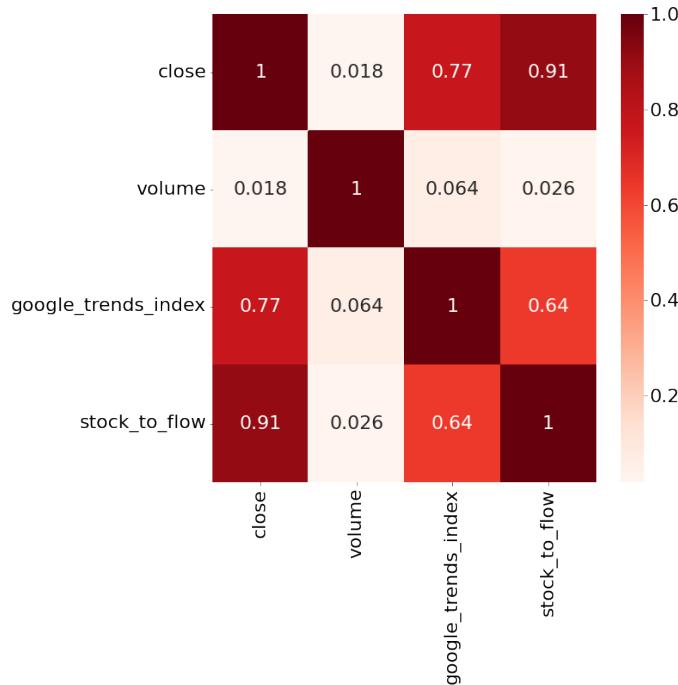


Figura 6.2: Esempio di matrice di correlazione.

Le metodologie di feature selection utilizzate sono state implementate in python. Per Boruta è stata sfruttata l'apposita libreria disponibile sulla repository GitHub *scikit-learn-contrib* [23]. La valutazione delle correlazioni di Pearson e Spearman è avvenuta mediante l'utilizzo del pacchetto *scipy.stats* [61].

#### 6.2.4 Feature selezionate

Prima di eseguire la feature selection è necessario definire accuratamente il problema di previsione. In particolare, trattandosi di un problema di apprendimento supervisionato, deve essere stabilita quale feature rappresenta l'etichetta di previsione  $y$ . Nel caso specifico di questa tesi, l'obiettivo è quello di prevedere il **prezzo di chiusura futuro di bitcoin**. Tutte le feature del dataset devono quindi essere valutate in base alla capacità predittiva e alla loro correlazione con il prezzo di chiusura di bitcoin. Tuttavia, è possibile che alcune feature siano direttamente influenzate dal prezzo e non viceversa. Ciò può portare facilmente in errore le metodologie di selezione, le quali potrebbero catalogare tali funzionalità come in forte correlazione, quando in realtà non forniscono ai modelli nessuna informazione predittiva. Nel tentativo di migliorare l'analisi, prima di effettuare la feature selection, la metrica relativa al prezzo di chiusura è stata **traslata** indietro di una riga. Trattandosi di un dataset con dati giornalieri, questo passaggio permette di analizzare la relazione delle funzionalità con il prezzo di chiusura del giorno successivo, enfatizzandone la capacità predittiva, e individuando reali anticipatori, a dispetto di metriche ridondanti.

Come visto nel Capitolo 3, gli indicatori tecnici finanziari sono calcolati utilizzando in modo diretto i valori di prezzo. Questo rende ovviamente correlate tali metriche al prezzo di chiusura, per questo motivo si è deciso di non sottoporre anche gli indicatori tecnici al processo di feature selection, ma di aggiungerle in seguito al dataset. Allo stesso modo non sono stati aggiunti i prezzi di apertura, chiusura, massimo, minimo e volume relativi alla giornata.

Boruta fornisce in output un "rank" corrispondente all'importanza che tale metodo riconosce alla feature. I valori del rank vanno da 1 a 30 (il valore 30 è dovuto a questo caso specifico, in quanto dipende dal numero totale di feature), dove 1 indica che la feature si trova nell'area di accettazione, e 30 rappresenta l'estremo opposto (ultima posizione dell'area di rifiuto). Per ridurre l'influenza della casualità (nel prendere i

sottoinsiemi di feature negli alberi della foresta), il metodo Boruta è stato eseguito 3 volte. Le caratteristiche di ogni esecuzione sono:

1. utilizzo di Random Forest come modello di apprendimento;
2. il numero di alberi della foresta è derivato automaticamente dal modello in base alla dimensione del dataset;
3. il numero di iterazioni massimo è 100. Nonostante ciò, se il modello riesce a determinare in anticipo il rank di ogni feature l'esecuzione termina prima;
4. la profondità massima raggiungibile, nello sviluppo di ogni albero varia in base alle tre esecuzioni: 5, 10 e una volta senza fornire alcun vincolo (l'albero viene esteso finché tutte le foglie non sono pure).

I risultati ottenuti sono stati analizzati e valutati con un sistema pesato, dando maggior importanza alle valutazioni espresse da Boruta (visto l'approccio più specifico sul problema). Lo schema di regole per assegnare pesi alle decisioni è stato definito assegnando dei colori a seconda dell'importanza delle feature. Vi sono tre possibili colori, ed ogni colore corrisponde ad un punteggio per la metrica. **Nel caso in cui il punteggio di una metrica sia superiore a 6.5, essa viene selezionata e aggiunta al dataset definitivo.** Lo schema per assegnare i colori è così definito:

- **Boruta**

- **verde** se  $rank = 1$ ;
- **giallo** se  $2 \leq rank \leq 5$ ;
- **arancione** se  $6 \leq rank \leq 12$ .

- **Pearson e Spearman**

- **verde** se il coefficiente  $|coef| \geq 0.85$ ;
- **giallo** se  $0.75 \leq |coef| < 0.85$ ;
- **arancione** se  $0.60 \leq |coef| < 0.75$ .

Lo schema dei **punteggi** in base al colore è il seguente:

- **Boruta**

- **verde** 5 punti;
- **giallo** se 2.5 punti;
- **arancione** se 1 punto.

- **Pearson e Spearman**

- **verde** 2 punti;
- **giallo** 1 punto;
- **arancione** 0.5 punti.

I valori per cui assegnare colori e punteggi sono stati scelti sulla base di diverse prove sperimentali effettuate. Con questo approccio si evita di rimuovere in modo drastico un numero eccessivo di feature. Gli intervalli di Pearson e Spearman sono stati scelti sulla base delle affermazioni riguardo alla forza della correlazione espresse in Sezione 6.2.3. Allo stesso modo, gli intervalli di importanza per Boruta, sono stati scelti in modo da rispettare le aree di selezione della distribuzione binomiale (vedi Figura 6.1). La Tabella 6.6 riporta i risultati ottenuti utilizzando Boruta nel dataset in cui la feature **close** (prezzo di chiusura) è stata traslata all'indietro. In questo modo sono evidenziate le caratteristiche predittive delle feature nei confronti del prezzo di chiusura del giorno successivo. Nella Tabella 6.7 e nella Tabella 6.8 sono riportati, per puro confronto, i risultati dell'esecuzione di Boruta utilizzando il dataset senza traslazione del prezzo di chiusura (permette di evidenziare l'importanza delle feature in relazione al prezzo di chiusura del giorno stesso). Nella tabella 6.9 e nella Tabella 6.10 sono riportati i punteggi ottenuti dalla correlazione di Pearson e dalla correlazione di Spearman utilizzando il dataset con il prezzo di chiusura traslato, in modo da evidenziare la correlazione delle feature con il prezzo di chiusura del giorno successivo. La Tabella 6.11 funge da sommario di tutti i risultati complessivi, ottenuti sul dataset con il prezzo di chiusura traslato, indicando quali feature sono state selezionate o meno.

FEATURE	BORUTA1	BORUTA2	BORUTA3
new addresses	23	24	21
active addresses	10	15	13
total unique addresses	1	1	1
sending addresses	24	29	28
receiving addresses	17	19	19
total fees	25	21	23
mean transaction fees	12	15	17
median transaction fees	20	26	25
number of transaction	27	28	30
transaction rate	27	27	27
market cap	1	1	1
median transfer volume	7	9	7
mean transfer volume	7	15	12
total transfer volume	22	20	18
google trends index	3	3	4
total transaction size	9	7	9
mean transaction size	16	17	19
hash rate	15	17	13
mining difficulty	5	5	5
reserve risk	1	1	1
stock to flow	1	1	1
stock to flow deflection	1	8	9
realized profit	13	11	15
realized loss	1	1	1
cvdd	1	1	1
address>001	18	30	28
addresses>01	1	1	1
addresses>1	1	1	1
addresses>10	2	2	2
addresses>100	3	3	3
addresses>1000	1	1	1
addresses>10k	10	13	9
exchanges withdrawls	6	6	5
exchanges deposits	14	10	11
exchage outflow	26	23	26
exchange inflow	1	1	1
mvr ratio	1	12	16
nvt signal	20	25	23

Tabella 6.6: Risultati Boruta sul dataset con prezzo di chiusura traslato.

Nella Tabella 6.6 sono riportati i risultati, dove:

- **BORUTA1** è l'esecuzione in cui non vi è il vincolo di profondità massima per gli alberi;
- **BORUTA2** è l'esecuzione in cui il vincolo di profondità massima per gli alberi è 5;
- **BORUTA3** è l'esecuzione in cui il vincolo di profondità massima per gli alberi è 10.

Per completezza, e per dimostrare la differenza dei risultati ottenuti nel caso in cui non venga effettuata la traslazione, qui di seguito sono riportate le valutazioni confrontando il prezzo di chiusura con le metriche relative allo stesso giorno.

FEATURE	BORUTA1	BORUTA2	BORUTA3
new addresses	23	24	21
active addresses	10	15	13
total unique addresses	1	1	1
sending addresses	24	29	28
receiving addresses	17	19	19
total fees	25	21	23
mean transaction fees	12	15	17
median transaction fees	20	26	25
number of transaction	27	28	30
transaction rate	27	27	27
market cap	1	1	1
median transfer volume	7	9	7
mean transfer volume	7	15	12
total transfer volume	22	20	18
google trends index	3	3	4
total transaction size	9	7	9
mean transaction size	16	17	19
hash rate	15	17	13
mining difficulty	5	5	5
reserve risk	1	1	1

Tabella 6.7: Risultati Boruta sul dataset con prezzo di chiusura non traslato. Parte 1.

FEATURE	BORUTA1	BORUTA2	BORUTA3
stock to flow	1	1	1
stock to flow deflection	1	8	9
realized profit	13	11	15
realized loss	1	1	1
cvdd	1	1	1
address>001	18	30	28
addresses>01	1	1	1
addresses>1	1	1	1
addresses>10	2	2	2
addresses>100	3	3	3
addresses>1000	1	1	1
addresses>10k	10	13	9
exchanges withdrawls	6	6	5
exchanges deposits	14	10	11
exchange outflow	26	23	26
exchange inflow	1	1	1
mvr ratio	1	12	16
nvt signal	20	25	23

Tabella 6.8: Risultati Boruta sul dataset con prezzo di chiusura non traslato. Parte 2.

I risultati ottenuti mediante l'applicazione della correlazione di Pearson e la correlazione di Spearman sono i seguenti (la Tabella 6.9 e la Tabella 6.10 rappresentano la colonna delle due matrici di correlazione relative al prezzo di chiusura):

FEATURE	PEARSON	SPEARMAN
new addresses	0,612	0,86
active addresses	0,673	0,889
total unique addresses	0,737	0,913
sending addresses	0,629	0,88
receiving addresses	0,625	0,863
total fees	0,179	0,388
mean transaction fees	0,006	-0,131
median transaction fees	-0,021	0,195

Tabella 6.9: Risultati Pearson e Spearman sul dataset con prezzo di chiusura traslato. Parte 1.

FEATURE	PEARSON	SPEARMAN
number of transaction	0,433	0,774
transaction rate	0,433	0,774
market cap	0,998	0,997
median transfer volume	-0,397	-0,87
mean transfer volume	-0,21	-0,69
total transfer volume	0,019	0,172
google trends index	0,773	0,903
total transaction size	0,574	0,852
mean transaction size	0,369	0,2
hash rate	0,785	0,913
mining difficulty	0,789	0,914
reserve risk	0,085	0,19
stock to flow	0,914	0,913
stock to flow deflection	-0,214	-0,27
realized profit	0,76	0,89
realized loss	0,495	0,818
cvdd	0,862	0,913
address>001	0,66	0,919
addresses>01	0,643	0,915
addresses>1	0,586	0,913
addresses>10	0,393	0,845
addresses>100	0,136	0,295
addresses>1000	0,645	0,689
addresses>10k	-0,256	0,293
exchanges withdrawls	0,691	0,871
exchanges deposits	0,589	0,868
exchage outflow	0,153	0,43
exchange inflow	0,141	0,421
mvr ratio	0,368	0,277
nvt signal	0,64	0,575

Tabella 6.10: Risultati Pearson e Spearman sul dataset con prezzo di chiusura traslato. Parte 2.

Utilizzando le regole sopra citate, dall'utilizzo di Boruta (con il dataset con il prezzo di chiusura traslato) e le due metodologie di selezione feature basate su filtro, si ottiene la seguente tabella:

FEATURE	B1	B2	B3	P	S	X
new addresses	0	0	0	0.5	2	2.5
active addresses	1	0	0	0.5	2	3.5
total unique addresses	5	5	5	0.5	2	17.5
sending addresses	0	0	0	0.5	2	2.5
receiving addresses	0	0	0	0.5	2	2.5
total fees	0	0	0	0	0	0
mean transaction fees	1	0	0	0	0	1
median transaction fees	0	0	0	0	0	0
number of transaction	0	0	0	0	1	1
transaction rate	0	0	0	0	1	1
market cap	5	5	5	2	2	19
median transfer volume	1	1	1	0	2	5
mean transfer volume	1	0	1	0.5	0	2.5
total transfer volume	0	0	0	0	0	0
google trends index	2.5	2.5	2.5	1	2	10.5
total transaction size	1	1	1	0	2	5
mean transaction size	0	0	0	0	0	0
hash rate	0	0	0	1	2	3
mining difficulty	2.5	2.5	2.5	1	2	10.5
reserve risk	5	5	5	0	0	15
stock to flow	5	5	5	2	2	19
stock to flow deflection	5	1	1	0	0	7
realized profit	0	1	0	1	2	4
realized loss	5	5	5	0	1	16
cvdd	5	5	5	2	2	19
address>001	0	0	0	0.5	2	2.5
addresses>01	5	5	5	0.5	2	17.5
addresses>1	5	5	5	0	2	17
addresses>10	2.5	2.5	2.5	0	1	8.5
addresses>100	2.5	2.5	2.5	0	0	7.5
addresses>1000	5	5	5	0.5	0.5	16
addresses>10k	1	0	1	0	0	2
exchanges withdrawls	1	1	2.5	0.5	2	7
exchanges deposits	0	1	1	0	2	4
exchage outflow	0	0	0	0	0	0
exchange inflow	5	5	5	0	0	15
mvr ratio	5	1	0	0	0	6
nvt signal	0	0	0	0.5	0	0.5

Tabella 6.11: Risultati complessivi feature selection.

dove su ogni cella è riportato il punteggio (in base al colore) ottenuto mediante la feature selection, e:

- **B1, B2, B3** sono le tre esecuzioni di Boruta;
- **P** corrisponde alla correlazione di Pearson;
- **S** corrisponde alla correlazione di Spearman;
- **X** indica se la feature è stata selezionata o meno.

Se il punteggio di ciascuna feature è  $\geq 6.5$ , la cella **X** è di colore blu, indicando che tale metrica è stata selezionata. Aggiungendo le feature relative agli indicatori tecnici finanziari e le metriche di prezzo, si ottiene un dataset con 29 feature: **open, high, low, close, volume, total unique addresses, market cap, google trends index, mining difficulty, reserve risk, stock to flow, stock to flow deflection, realized loss, cvdd, addresses>01, addresses>1, addresses>10, addresses>100, addresses>1000, exchanges withdrawls, exchange inflow, SMA<sub>50</sub>, SMA<sub>200</sub>, EMA<sub>12</sub>, EMA<sub>26</sub>, MACD, KD, RSI, WR<sub>14</sub>**.

La Tabella 6.12 riporta le feature selezionate, ordinate in base all'importanza da 1 a 3, dove 1 indica le feature più rilevanti (sono stati scelti tre valori di importanza in modo da distinguere meglio le feature). Per le metriche aggiunte successivamente al dataset (dopo aver eseguito feature selection) il punteggio non è disponibile in quanto risultano ovviamente fortemente correlate (non sono state integrate nel processo di feature selection per non influenzare la distribuzione binomiale e le aree di selezione di Boruta). La sigla "N.D" in Tabella 6.12 indica che il punteggio su tale metrica non è stato calcolato, in quanto non è stata processata mediante le tecniche di feature selection (Boruta, Pearson e Spearman), ma aggiunta direttamente al dataset.

FEATURE	IMPORTANZA	PUNTEGGIO
<b>open</b>	1	N.D.
<b>high</b>	1	N.D.
<b>low</b>	1	N.D.
<b>close</b>	1	N.D.
<b>volume</b>	1	N.D.
<b>SMA<sub>50</sub></b>	1	N.D.
<b>SMA<sub>200</sub></b>	1	N.D.
<b>EMA<sub>12</sub></b>	1	N.D.
<b>EMA<sub>26</sub></b>	1	N.D.
<b>MACD</b>	1	N.D.
<b>KD</b>	1	N.D.
<b>RSI</b>	1	N.D.
<b>WR<sub>14</sub></b>	1	N.D.
<b>stock to flow</b>	1	19
<b>cvdd</b>	1	19
<b>market cap</b>	1	19
<b>total unique addresses</b>	2	17.5
<b>addresses&gt;01</b>	2	17.5
<b>addresses&gt;1</b>	2	17
<b>realized loss</b>	2	16
<b>addresses&gt;1000</b>	2	16
<b>reserve risk</b>	2	15
<b>exchange inflow</b>	2	15
<b>google trends index</b>	3	10.5
<b>mining difficulty</b>	3	10.5
<b>addresses&gt;10</b>	3	8.5
<b>addresses&gt;100</b>	3	7.5
<b>stock to flow deflection</b>	3	7
<b>exchange withdrawals</b>	3	7

Tabella 6.12: Tabella feature selezionate in ordine di importanza.

### 6.3 Pre-processing dei dati

Solitamente, prima di passare all’allenamento e al test dei modelli, risulta necessario eseguire un’ulteriore fase di pre-elaborazione dei dati. In particolare, il dataset generale deve essere suddiviso in due set di dati: uno per l’**allenamento** e uno per la

successiva fase di **test** del modello. Oltre a ciò, può esservi la necessità di organizzare i dati in opportune strutture dati prima di passarle in input ai modelli, come nel caso di reti neurali LSTM (vedi Sezione 6.3.2). Alcuni modelli di apprendimento automatico basati sulla distanza euclidea, discesa del gradiente o altri approcci che dipendono da misure di prossimità (come ad esempio modelli KNN, SVM e reti neurali multilivello), sono particolarmente influenzati dai valori degli attributi del dataset. Nel caso in cui i valori delle feature non siano tutti sulla stessa scala, il modello può facilmente essere portato in errore. Questo perché, involontariamente, viene dato un peso decisionale maggiore ad alcune metriche rispetto ad altre, rendendo la predizione del modello fortemente dominata da esse. In questi casi è necessario effettuare uno **scaling** dei dati, portando sulla stessa scala tutti i valori delle feature, in modo che ognuna contribuisca in modo proporzionato alla decisione finale. Le principali tecniche di scaling, nonché quelle utilizzate, e la preparazione dei dati per i rispettivi modelli, sono descritte nelle sezioni seguenti.

### 6.3.1 Normalizzazione e Standardizzazione

Molto spesso, come in questo specifico problema, può capitare che il range di valore delle metriche del dataset non sia lo stesso. Prendendo un esempio, le istanze della feature **google trends index** possono assumere valori tra 0 e 100, mentre quelle relative alla metrica **market cap** operano in un valore tra 0 e oltre 1 triliardo. Tale discrepanza di scala porta facilmente in errore i modelli di previsione. Per risolvere questa problematica, esistono diverse tecniche di ridimensionamento dei dati, le quali permettono di portare tutti i valori sulla stessa scala:::

- **Normalizzazione.** Chiamata anche **Min-Max Scaling**, è un metodo in cui i dati vengono ridimensionati su un intervallo fisso. La versione classica riduce i dati nell'intervallo tra 0 e 1, si ottiene con la seguente formula:

$$x_i \text{ normalizzato} = \frac{x_i - \min(x)}{\max(x) - \min(x)}, \quad (6.1)$$

dove  $\max(x)$  e  $\min(x)$  corrispondono rispettivamente al massimo valore e al minimo valore per quella feature. In alcuni casi, può essere preferibile avere i valori

nell’intervallo  $[-1, 1]$ , ciò è possibile utilizzando la **normalizzazione media**:

$$x_i \text{ normalizzato} = \frac{x_i - \text{media}(x)}{\max(x) - \min(x)}.$$

L’unica differenza rispetto alla Formula 6.1, è che al numeratore, al posto del minimo valore ( $\min(x)$ ) vi è la media dei valori della feature x (cioè  $\text{media}(x)$ ).

- **Standardizzazione.** È il processo di ridimensionamento di uno o più attributi in modo che abbiano un valore medio di 0 e una deviazione standard di 1. Le caratteristiche sono quindi ridimensionate in modo da avere le proprietà di una distribuzione normale con media nulla e deviazione standard pari a 1. Viene anche chiamata **normalizzazione Z-Score**, in quanto ogni valore del dataset viene trasformato in un **valore Z**, così definito:

$$Z_i = \frac{x_i - u}{\sigma},$$

dove  $u$  è la media dei campioni di addestramento,  $\sigma$  la deviazione standard e  $x_i$  è il valore da standardizzare.

La normalizzazione è utile quando, a priori, si ha la conoscenza che la distribuzione dei dati non segue una distribuzione gaussiana. Può essere utile in algoritmi che non presuppongono alcuna distribuzione dei dati come K-Nearest Neighbors e reti neurali. La standardizzazione, d’altra parte, può essere utile nei casi in cui i dati seguono una distribuzione gaussiana. Tuttavia, ciò non deve essere necessariamente vero. Inoltre, a differenza della normalizzazione, la standardizzazione non ha un intervallo di delimitazione, e nonostante vi siano valori anomali, i dati non sono influenzati negativamente dalla standardizzazione [7]. A seconda delle caratteristiche del problema, del modello di apprendimento e dei dati, può essere utilizzata una tecnica piuttosto che un’altra. Non esiste una regola rigida che determina quando normalizzare o standardizzare i dati. È sempre consigliato implementare i modelli adattando sia i dati grezzi, che normalizzati e standardizzati, confrontare le performance e scegliere la tecnica che permette di ottenere i risultati migliori. Qui di seguito sono descritte le metodologie di ridimensionamento dati utilizzate (quando necessario) nell’implementazione dei modelli.

**Random Forest.** Nell’implementazione di questo modello non state necessarie tecniche di normalizzazione e standardizzazione dei dati. Il modello di apprendimento Random Forest non si basa su tecniche di comparazione che dipendono dai valori dei record del dataset. La creazione degli alberi decisionali interni alla foresta si basa solo sulla valutazione degli split, ottenuti dalle test condition ad ogni nodo. I valori delle feature, anche se fuori scala tra di loro, non influenzano le prestazioni del modello [7].

**SVM.** Come già precisato, nell’implementazione di modelli basati su misure di prossimità o distanza euclidea, è necessario effettuare una tecnica di *scaling* dei dati. Le performance delle macchine a vettori di supporto, essendo fortemente influenzate dalla scelta dei punti con cui definire la posizione dell’iperpiano, hanno bisogno che i dati siano sulla stessa scala. Nell’implementare i modelli SVM, entrambe le metodologie di ridimensionamento sono state applicate, e le prestazioni a seconda dell’utilizzo di normalizzazione o standardizzazione sono riportate nella Sezione 6.6.

**LSTM.** Allo stesso modo dei modelli SVM, le prestazioni delle reti neurali LSTM sono particolarmente influenzate e danneggiate dalla presenza di dati fuori scala. Entrambe le tecniche di normalizzazione e standardizzazione sono state utilizzate, ne sono state analizzate le performance, e la metodologia da cui derivano prestazioni migliori è stata utilizzata per i test successivi (vedi Sezione 6.7).

Sia la normalizzazione che la standardizzazione, quando utilizzate, devono essere adattate sul solo set di dati di allenamento. Il modello di trasformazione ottenuto è successivamente utilizzato per riportare i dati di allenamento e test in scala. La mappatura non viene effettuata utilizzando tutto il dataset generale perchè la scala verrebbe influenzata anche dai dati di test, quando nel caso reale il modello non ha a disposizione i valori di quest’ultimi.

Lo *scaling* dei dati è stato effettuato utilizzando l’apposito pacchetto di funzioni “*preprocessing*” (**StandardScaler** e **MinMaxScaler**) fornito dalla libreria python *sklearn* [55].

### 6.3.2 Dati di training e test

In questa sezione vengono descritte le fasi di elaborazione dei set di dati utilizzati dai vari modelli per l'allenamento e per il successivo test. Trattandosi in tutti e tre i casi di modelli ad apprendimento supervisionato, è necessario definire il set di record **X** e il set di *labels* **y**.

**Random Forest.** I dati di allenamento, per i modelli random forest, non sono stati aggregati in strutture dati particolari. Sia nel caso del modello di regressione, che nel caso del modello di classificazione, l'insieme dei record **X** corrisponde semplicemente al dataset di origine. Avendone la possibilità, sono state testate e confrontate le performance con entrambe le versioni del set di dati, sia utilizzando tutte le metriche a disposizione, sia utilizzando le sole metriche ottenute dal processo di feature selection (vedi Sezione 6.5). L'insieme di etichette **y**, invece, differisce a seconda del modello utilizzato. Nel caso della regressione, esso corrisponde alla colonna del dataset relativa alla metrica **close** (prezzo di chiusura del giorno) traslata di un valore  $n$ , che varia a seconda di quanto la previsione deve essere effettuata nel futuro ( $n = 1$  se si prevede il prezzo del giorno successivo,  $n = 7$  se si prevede il prezzo a 7 giorni nel futuro). Nel modello classificatore, è stato necessario rielaborare il valore **close** (traslato) in delle etichette categoriche. Le etichette sono: **buy**, **sell**, **hold**, definite secondo lo schema riportato nella Sezione 6.4. Le etichette di classe sono state successivamente processate utilizzando un *LabelEncoder*, il quale permette di convertire le classi, sotto forma di valore testuale, in etichette di destinazione con un valore compreso tra 0 e  $n\_classi - 1$  ( $buy = 0$ ,  $hold = 1$ ,  $sell = 2$ ).

**SVM.** La preparazione dei set **X** e **y** in questo modello è la stessa effettuata per le random forests, con la differenza che i dati sono precedentemente sottoposti allo *scaling*.

**LSTM.** L'input di una rete neurale LSTM differisce particolarmente dai modelli sopracitati. L'input di ogni livello LSTM (sia per un modello di classificazione che per un modello di regressione) è **tridimensionale**. Le tre dimensioni di input sono [11]:

- **samples** o campione;
- **time steps** o passi temporali;

- **feature.**

Un campione corrisponde ad un’istanza di dati con la stessa funzione dei record nei modelli SVM e random forest. A differenza dei record (che corrispondono ad array monodimensionali di lunghezza uguale al numero di feature), un campione è caratterizzato da 2 dimensioni: passi temporali e feature. Il valore time steps indica quanti record della serie temporale sono passati ad ogni iterazione. Ad esempio, l’input per un livello LSTM può essere definito da ( $\text{samples} = 2000$ ,  $\text{timesteps} = 7$ ,  $\text{feature} = 29$ ), indicando che il set **X** deve essere composto da:

- 2000 array (samples)
- ciascuno costituito da 7 array (time steps)
- di lunghezza 29 (feature).

Con questa configurazione ad ogni passaggio (2000 passaggi totali) il modello lavora su una serie temporale di 7 record (ogni riga del dataset è processata insieme alle 6 precedenti). Ciò significa che il livello di input si aspetta un array a tre dimensioni di dati. Nell’implementazione dei modelli LSTM sono state provate diverse possibili configurazioni per l’input (vedi Sezione 6.7). La definizione del set **y** differisce in base al tipo di modello (classificatore e regressore). Nel caso del modello di regressione le etichette sono semplicemente i valori numeri traslati di  $n$  giorni, come visto anche per SVM e random forest. Per il modello classificatore, le etichette categoriche, calcolate secondo lo schema **buy**, **sell**, **hold**, sono state trasformate nella codifica *one-hot*.

I set **X** e **y** per il test dei modelli sono stati definiti con gli stessi procedimenti utilizzati per i set di allenamento.

## 6.4 Sistema di simulazione di trading

Per valutare i modelli, è stato implementato un sistema che va ad effettuare delle simulazioni di trading, secondo le previsioni e le decisioni da essi fornite. In modo da avere un rapporto quanto più veritiero possibile, i test sono svolti solamente negli ultimi 3 anni. La simulazione verte nell’emulare un vero e proprio utilizzo del modello, che avrebbe operato giornalmente in un periodo continuato, con valori di prezzo reali

e commissioni di acquisto e vendita.

Dati e configurazione iniziale:

- ogni modello opera partendo con un portafoglio in dollari dal valore di 100'00\$;
- ogni operazione effettuata ha un costo dello 0,1% rispetto al valore della transazione (sia nel caso di acquisto o vendita di bitcoin);
- per ridurre il rischio, ogni operazione utilizza il 10% del portafoglio. Nel caso di un acquisto, viene utilizzato il 10% del portafoglio in dollari a disposizione. Nel caso di una vendita, viene utilizzato il 10% del portafoglio in bitcoin a disposizione;
- non si può procedere ad una vendita se il saldo del portafoglio bitcoin è a 0.

Il modello ha quindi il compito di individuare un momento opportuno per effettuare il primo acquisto. Solo nel momento in cui verrà previsto che il prezzo futuro di bitcoin salirà, rispetto al prezzo del momento in cui viene effettuata la previsione, verrà effettuato un acquisto.

Il profitto viene calcolato in base alla differenza percentuale di valore del portafoglio (calcolata in dollari). Il valore sarà dato dalla somma dei dollari e del controvalore in dollari dei btc in portafoglio. Il valore viene calcolato il giorno della fine del periodo di test.

Ogni **classificatore** restituisce in output un'etichetta, la quale rappresenta la decisione di trading che il sistema dovrà prendere. Le possibili etichette sono **buy**, **sell** e **hold**:

- se il sistema prevede un aumento di prezzo di oltre lo 0,1% (ovvero la commissione da pagare nel caso si effettui un'operazione), restituirà l'etichetta **buy**;
- se prevede una discesa del prezzo di oltre lo 0,1% restituirà l'etichetta **sell**;
- nel caso in cui la variazione di prezzo prevista sia minore dello 0,1% (sia in aumento che diminuzione), il sistema restituirà **hold**, e nessuna operazione di compravendita verrà eseguita.

Ogni **regressore**, invece, restituisce in output il valore di prezzo previsto. Sarà poi il sistema a valutare la variazione percentuale, rispetto al giorno della previsione, e ad effettuare un'operazione di acquisto, vendita o hold (seguendo lo stesso schema del classificatore). Si noti che, in questo modello, la previsione viene fatta direttamente sul prezzo, e l'etichetta viene calcolata successivamente.

Ogni modello opera su un intervallo di tempo giornaliero, le previsioni sono effettuate ogni giorno rispetto al prezzo di  $n$  giorni nel futuro (il valore di  $n$  è stato testato su 1, 7, 14 e 30, come verrà mostrato nelle sezioni successive). Per determinare il valore migliore per  $n$ , vengono effettuate molteplici simulazioni su periodi casuali, secondo il seguente schema:

- 10 simulazioni su intervalli casuali di 30 giorni;
- 10 simulazioni su intervalli casuali di 90 giorni;
- 3 simulazioni su intervalli casuali di 180 giorni;
- 1 simulazione su un intervallo casuale di 365 giorni;
- 1 simulazione su un intervallo casuale di 730 giorni.

Ogni periodo di test è un intervallo dove il punto iniziale è scelto casualmente negli ultimi 800 giorni. Per preservare l'affidabilità del modello, **trattandosi di un problema su serie temporali**, i dati, di allenamento e test, sono processati in modo continuo e in ordine cronologico (non ci sono salti e buchi temporali e i dati di allenamento sono sempre cronologicamente precedenti a quelli di test). Ad esempio se il periodo di test è nell'intervallo *01/05/2019 - 01/05/2020*:

- Il sistema è allenato su tutti i dati (in ordine cronologico) dall'inizio del dataset, fino al giorno *01/05/2019*;
- prima di ogni previsione, viene aggiunto un record al set di allenamento (il record relativo al giorno stesso) e il modello viene riallenato. Se oggi il modello effettua una previsione sul prezzo a 7 giorni nel futuro, il modello è stato allenato con un dataset in cui l'ultimo record contiene i dati odierni. Il giorno seguente, prima di effettuare una nuova previsione, il modello sarà riallenato con tutto il dataset precedentemente usato più un nuovo record, ovvero i dati di tale giorno.

- Tutti i dati a seguito del giorno 01/05/2020 non saranno nè testati, nè utilizzati per l’allenamento dal sistema.

In questo modo le proprietà della serie temporale sono preservate. Le performance ottenute dai modelli, sui diversi  $n$ , sono successivamente analizzate e, una volta determinato il valore  $n$  per cui si hanno i risultati migliori (per ogni modello), vengono effettuate delle simulazioni sui seguenti intervalli: ultimo anno, ultimi 2 anni e ultimi 3 anni. Quest’ultime permettono di avere un rendiconto di come il sistema avrebbe agito e performato in un periodo continuo e prolungato. Ogni simulazione è confrontata con una strategia di tipo *buy and hold*, in cui si investe da subito l’intero ammontare del portafoglio, permettendo di avere una valutazione rispetto all’andamento del mercato.

Per lo sviluppo dell’intera ricerca è stato utilizzato Google Colab [26]. Nello specifico è stato sottoscritto un abbonamento Pro+ [27], il quale permette di usufruire di GPU dedicate, fornite da Google. Ciò ha permesso di ridurre sensibilmente i tempi di allenamento e simulazione, su modelli relativamente complessi come reti neurali LSTM. Sono inoltremesse esecuzioni in background, le quali si sono rivelate fondamentali nel caso di simulazioni che hanno impiegato oltre 12 ore per la loro realizzazione. Per ogni simulazione è riportato il tipo di GPU utilizzato e il tempo di esecuzione necessitato.

## 6.5 Random Forest

In questa sezione, sono riportati i risultati dei test utilizzando modelli di apprendimento automatico Random Forest. Nella Sezione 6.5.1 sono descritti i test effettuati con un modello di classificazione, testando i possibili valori di  $n$ . Determinato il miglior  $n$ , esso è stato utilizzato per tutti i test successivi del modello. Sono inoltre mostrate le differenze allenando il modello di classificazione con entrambi i dataset a disposizione (dataset completo e dataset ottenuto mediante feature selection). Il dataset con cui si hanno le prestazioni migliori sarà poi utilizzato per tutti i successivi test dei modelli. Nella Sezione 6.5.2 sono riportati i risultati del modello di regressione. Nella Sezione 6.5.3 sono infine analizzati i test effettuati su time frime di ampio termine, confrontando le performance dei modelli con l’approccio *buy and hold*. Per la classificazione è stato utilizzato il modello *sklearnensemble.RandomForestClassifier*,

mentre per la regressione il modello `sklearn.ensemble.RandomForestRegressor` [55]. Le configurazioni dei modelli sono le seguenti:

- `n_estimators`: 100;
- `criterion`: *gini*;
- `max_depth`: None;
- `min_samples_split`: 2;
- `min_samples_leaf`: 1.

### 6.5.1 Classificatore

L’obiettivo primario di questa fase di test è determinare il parametro  $n$ . Per avere una metrica di giudizio quanto più veritiera possibile, è stata eseguita una simulazione (secondo lo schema riportato nella Sezione 6.4). Le performance con  $n > 30$  decrementano velocemente quindi non sono state prese in considerazione. Nella Tabella 6.13 sono riportati i risultati ottenuti utilizzando il dataset ricavato dalla feature selection in Sezione 6.2.1 (mediati in base al numero di test effettuati per intervallo). Le migliori performance sono ottenute con un valore di  $n$  pari a 30. Ciò potrebbe essere spiegato dal fatto che Bitcoin risulta essere un asset altamente volatile. Ciò rende di difficile previsione l’andamento di prezzo del giorno immediatamente successivo, in quanto repentine e continue oscillazioni si ripercuotono sul mercato. È semplice verificare che nonostante una fase di uptrend (sul macro periodo), il prezzo del giorno successivo potrebbe subire correzioni, e viceversa, in un downtrend il prezzo del giorno successivo potrebbe comunque rimbalzare in dei livelli di supporto prima di continuare la decrescita. Questo, in aggiunta all’estrema volatilità potrebbe portare il modello di previsione in errore e occorrere in performance sul portafoglio negative. Il modello di apprendimento automatico riesce, invece, ad identificare con maggior precisione il macro andamento di prezzo in un time frame a 30 giorni, ignorando variazioni di prezzo intermedie. Per avere un’ulteriore metrica di valutazione, la simulazione con  $n = 30$  è stata ripetuta raddoppiando il numero di test per ogni intervallo (vedi Tabella 6.14). Nella Tabella 6.15 sono riportati i risultati ottenuti utilizzando il dataset completo in una previsione con  $n = 30$ . La GPU utilizzata

per le simulazioni è: NVIDIA-SMI 470.74 Driver Version: 460.32.03 CUDA Version: 11.2. Il tempo di esecuzione richiesto varia a seconda del dataset utilizzato e del tipo di test. Per la simulazione in Tabella 6.13 sono stati necessari 180 minuti circa. La simulazione in Tabella 6.14 ha impiegato circa 90 minuti. La simulazione in Tabella 6.15 ha impiegato circa 110 minuti.

$n = 1$			
intervallo (in giorni)	test	accuracy	profitto
30	10	41.00%	+4.81%
90	10	41.11%	+1.00%
180	3	43.51%	+12.86%
365	1	44.93%	+38.52%
730	1	43.15%	+102.81%
$n = 7$			
intervallo (in giorni)	test	accuracy	profitto
30	10	47.00%	+6.08%
90	10	51.33%	+24.89%
180	3	47.96%	+5.91%
365	1	53.42%	+246.54%
730	1	49.86%	+269.12%
$n = 14$			
intervallo (in giorni)	test	accuracy	profitto
30	10	61.00%	+8.16%
90	10	59.00%	+14.41%
180	3	57.70%	+46.20%
365	1	57.80%	+220.51%
730	1	54.37%	+317.26%
$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	10	77.66%	+15.06%
90	10	63.66%	+28.87%
180	3	57.59%	+101.67%
365	1	62.46%	+76.92%
730	1	60.82%	+808.68%

Tabella 6.13: Risultati simulazione multi-periodo su modello classificatore Random Forest con l'obiettivo di determinare il miglior valore di  $n$ .

$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	20	65.83%	+9.95%
90	20	71.05%	+40.92%
180	6	57.03%	+63.51%
365	2	58.76%	+72.50%
730	2	62.19%	+920.23%

Tabella 6.14: Risultati simulazione multi-periodo su modello classificatore Random Forest prevedendo l’andamento del prezzo a 30 giorni nel futuro, utilizzando il dataset con feature selezionate in Sezione 6.2.4.

La tecnica di feature selection utilizzata nella Sezione 6.2.1 opera nel caso in cui la previsione venga effettuata per il giorno successivo. Per questo motivo, avendo individuato il time frame a 30 giorni come intervallo di previsione migliore, è necessario riapplicare la feature selection. I risultati ottenuti dalla nuova feature selection indicano che tutte le metriche del dataset generale rientrano nell’area di accettazione. È stata eseguita una nuova simulazione del modello di classificazione random forest con  $n = 30$ , utilizzando, per allenamento e test, tutte le metriche a disposizione (vedi Tabella 6.15). Avendo a disposizione una mole maggiore di dati, i tempi di esecuzione sono stati maggiori. Nonostante ciò, non sono distinguibili immediatamente notevoli migliorie nelle performance. Tuttavia, confrontando un intervallo continuo di 1 anno, utilizzando o meno tutte le feature, risulta evidente un incremento delle prestazioni (vedi Sezione 6.5.3 per dimostrazione).

$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	20	60.66%	+8.21%
90	20	58.22%	+15.92%
180	6	60.00%	+85.98%
365	2	55.20%	+58.55%
730	2	62.73%	+805.34%

Tabella 6.15: Risultati simulazione multi-periodo su modello classificatore Random Forest prevedendo l’andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili.

### 6.5.2 Regressore

Appurato che il miglior valore per  $n$  è 30, e il dataset da utilizzare in questo caso risultò essere quello completo, le simulazioni per il modello di regressione sono state effettuate con questi parametri. Trattandosi di una regressione, non è possibile misurare le performance utilizzando l'accuracy. Per avere una comparazione con il classificatore è stata riportata solo la percentuale media di profitto ottenuta su ogni singolo periodo. Per quanto riguarda le ulteriori metriche di prestazione, del regressore, sono riportate in Sezione 6.5.3. In questo caso non è stato possibile utilizzare la GPU fornita da Google Colab in quanto i limiti di utilizzo sono stati precedentemente raggiunti con gli altri test. Il tempo di esecuzione richiesto è stato di circa 8 ore. I tempi di esecuzione sono stati notevolmente maggiori, nonostante ciò le performance ottenute risultano peggiori (vedi Tabella 6.16).

n = 30		
intervallo (in giorni)	test	profitto
30	20	+2.74%
90	20	+10.90%
180	6	+17.61%
365	2	+47.77%
730	2	+161.00%

Tabella 6.16: Risultati simulazione multi-periodo su modello regressore Random Forest prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili.

### 6.5.3 Analisi dei modelli su time frame ampio

In questa sezione, sono riportati i risultati relativi alle simulazioni su una serie temporale prolungata, in modo da avere un riscontro effettivo di come i modelli avrebbero performato in una reale applicazione. Le simulazioni sono state effettuate su 1 anno, 2 anni e 3 anni. Il modello opera su una previsione a 30 giorni nel futuro in ogni simulazione. Per la simulazione ad 1 anno sono state confrontate le prestazioni del modello di classificazione utilizzando il dataset ottenuto dalla feature selection (per  $n = 1$ ) e utilizzando il dataset completo (ovvero dataset ottenuto con feature selection con  $n = 30$ ). La comparazione è effettuata solo su tale periodo, per tutti gli intervalli successivi viene preso in considerazione solo il modello migliore (ovvero utilizzando

tutto il dataset). Il regressore è invece testato direttamente mediante il dataset completo. Tutte le simulazioni sono confrontate con un approccio **buy and hold**. Per ogni simulazione, sono disponibili tutte le metriche di valutazione per i modelli. In ogni periodo quindi verrà riportato:

- prestazioni e valore portafoglio dopo una strategia di holding, investendo il 100% del portafoglio il primo giorno;
- prestazioni e valore del portafoglio dopo la strategia di trading del nostro sistema, basata sulle previsioni di quest'ultimo;
- le performance e le previsioni del modello.

### SIMULAZIONE 1 ANNO:

- **inizio:** 2020-05-26;
- **fine:** 2021-05-26;
- **prezzo iniziale BTC** 8830.00\$;
- **prezzo finale BTC** 39266.35\$.

In Tabella 6.17 sono riportate le performance dei classificatori implementati, comparati alla strategia di buy and hold. Si può notare che utilizzando il dataset con tutte le metriche (risultante anche se si effettua feature selection con traslazione per  $n = 30$ ) le prestazioni sono effettivamente migliori. Nonostante ciò, nessuno dei modelli riesce a battere il mercato (strategia buy and hold). I risultati sono comunque positivi, si può notare infatti una minor volatilità del portafoglio rispetto al mercato (fattore ritenuto essenziale negli investimenti). Un altro fattore fin da subito rilevabile riguarda le prestazioni del regressore. Quest'ultime risultano essere nettamente inferiori rispetto ai modelli di classificazione. Un'analisi accurata del problema di regressione sarà presentata a fine sezione.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
<b>strategia buy and hold</b>	444247.47\$	+344.24%
<b>classificazione utilizzando il dataset ottenuto da feature selection</b>	279921.24\$	+179,92%
<b>classificazione utilizzando il dataset completo</b>	305100.00\$	+205,10%
<b>regressione utilizzando il dataset completo</b>	135945.75\$	+35,94%

Tabella 6.17: Comparazione performance su simulazione ultimo anno.

In Figura 6.3, Figura 6.4 e Figura 6.5, sono riportate le prestazioni dei portafogli d’investimento utilizzando le diverse strategie (strategia buy and hold, modello di classificazione e modello di regressione). Le prestazioni del modello, utilizzando il dataset con meno feature, sono inferiori al corrispettivo modello utilizzando tutte le metriche a disposizione. Per questo motivo, i modelli di apprendimento automatico basati su foreste casuali, non sono ulteriormente testati sul dataset ottenuto in Sezione 6.2.4. Tutte le simulazioni successive sono effettuate utilizzando tutte le metriche a disposizione. In Figura 6.6 e Figura 6.7 sono riportate graficamente le previsioni dei modelli (classificatore e regressore). In Tabella 6.18, Tabella 6.19 e Tabella 6.20, sono riportate le metriche di valutazione per i diversi modelli di apprendimento automatico).

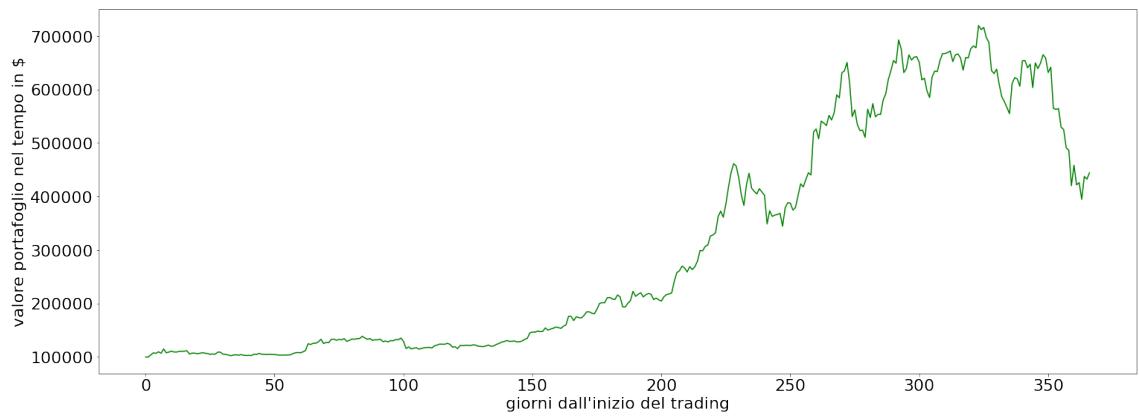


Figura 6.3: Performance portafoglio su un anno con strategia buy and hold.

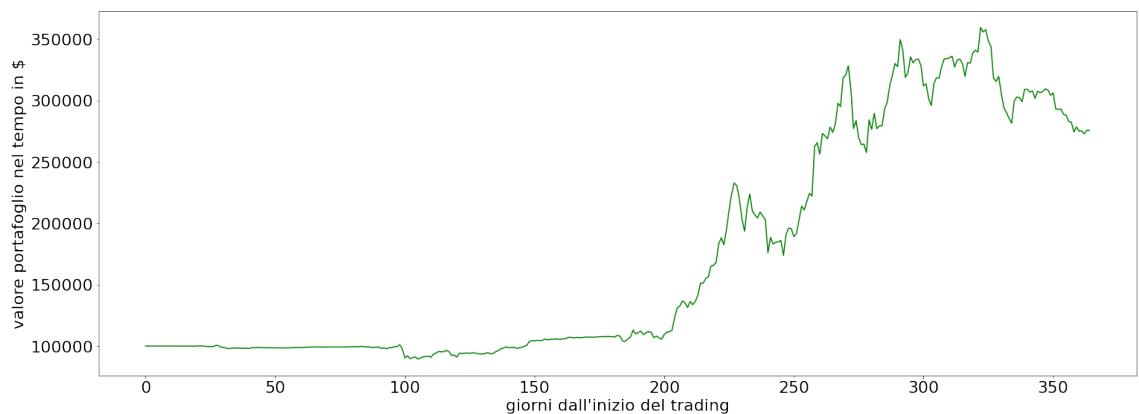


Figura 6.4: Performance portafoglio su un anno, utilizzando il modello random forest di classificazione (dataset completo, ricavato con feature selection con  $n = 30$ ).

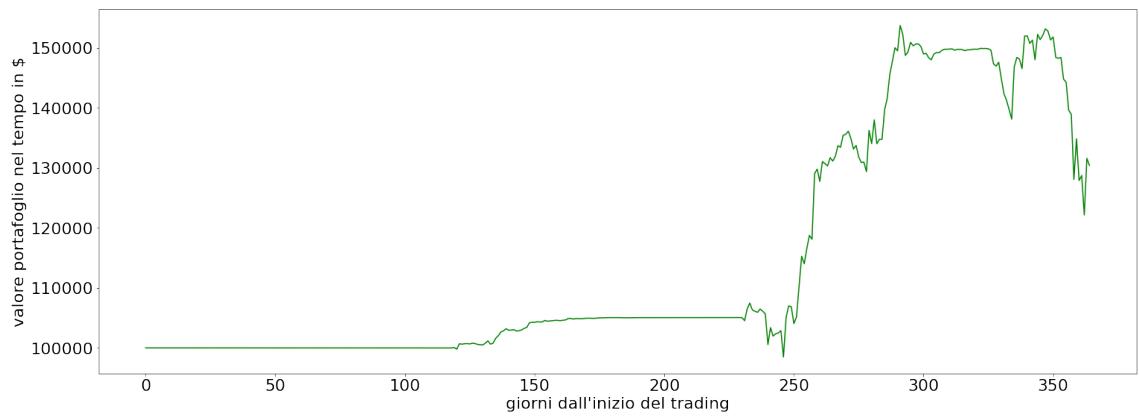


Figura 6.5: Performance portafoglio su un anno, utilizzando il modello random forest di regressione.

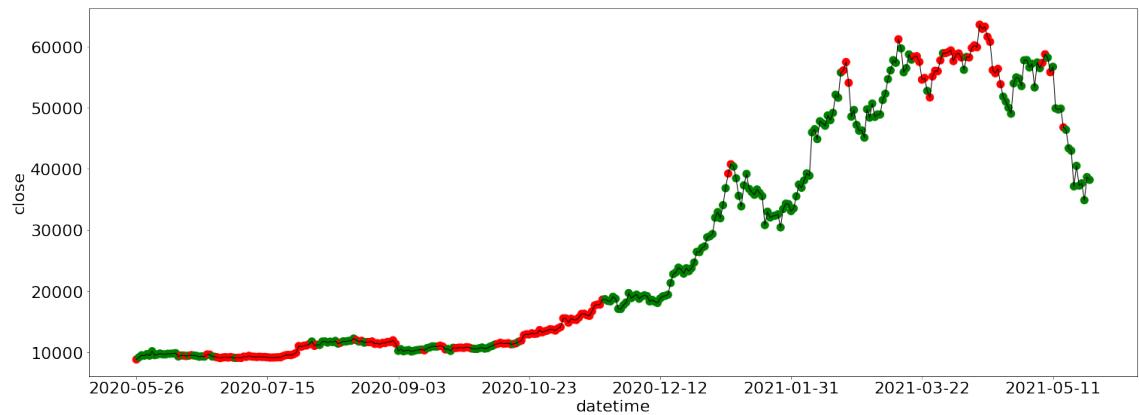


Figura 6.6: Previsioni del modello di classificazione random forest su un anno, utilizzando il dataset completo. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

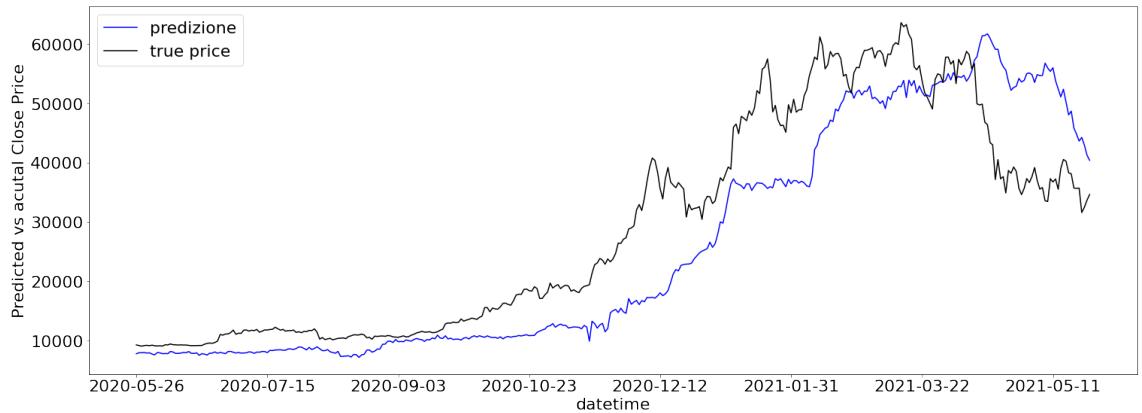


Figura 6.7: Previsioni del modello di regressione random forest su un anno, utilizzando il dataset completo. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	68%	65%	66%	235
<b>1 (hold)</b>	0%	0%	0%	2
<b>2 (sell)</b>	41%	45%	43%	128
<b>accuracy del modello: 57.53%</b>				

Tabella 6.18: Performance del modello classificatore random forest su un anno di test utilizzando il dataset ottenuto mediante feature selection in Sezione 6.2.4.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	70%	65%	67%	235
<b>1 (hold)</b>	1%	0%	0%	2
<b>2 (sell)</b>	44%	50%	47%	128
<b>accuracy del modello: 59.17%</b>				

Tabella 6.19: Performance del modello classificatore random forest su un anno di test utilizzando il dataset con tutte le metriche.

PERFORMANCE	
<b>Mean Absolute Error (MAE)</b>	7274.548717933222
<b>Mean Squared Error (MSE)</b>	86849227.40126932
<b>Root Mean Squared Error (RMSE)</b>	9319.293288724704
<b>Max Error</b>	23432.530049946297
$R^2$	0.7331538964489407

Tabella 6.20: Performance del modello regressore random forest su un anno di test utilizzando il dataset con tutte le metriche.

### SIMULAZIONE 2 ANNI:

- **inizio:** 2019-05-27;
- **fine:** 2021-05-26;
- **prezzo iniziale BTC** 8802.75\$;
- **prezzo finale BTC** 39266.35\$.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
<b>strategia buy and hold</b>	445622.70\$	+345.62%
<b>classificazione utilizzando il dataset completo</b>	503752.98\$	+403,75%
<b>regressione utilizzando il dataset completo</b>	212917.91\$	+112.91%

Tabella 6.21: Comparazione performance su simulazione ultimi 2 anni.

In Figura 6.8, Figura 6.9 e Figura 6.10, sono riportate le prestazioni dei portafogli utilizzando le diverse strategie. In Figura 6.11 e Figura 6.12, sono riportate graficamente le previsioni dei modelli. In Tabella 6.22 e Tabella 6.23, sono riportate le metriche

di valutazione per i diversi modelli di apprendimento automatico. A differenza della simulazione su un singolo anno, in questo caso, il classificatore riesce a battere le performance di una strategia **buy and hold**. Il modello è riuscito a prevedere l'estrema salita di prezzo avvenuta nella prima metà del 2021. Ciò ha permesso al modello di sfruttare il forte uptrend e di andare in profitto, vendendo gran parte dei btc in portafoglio, prima del forte crollo di Maggio 2021.

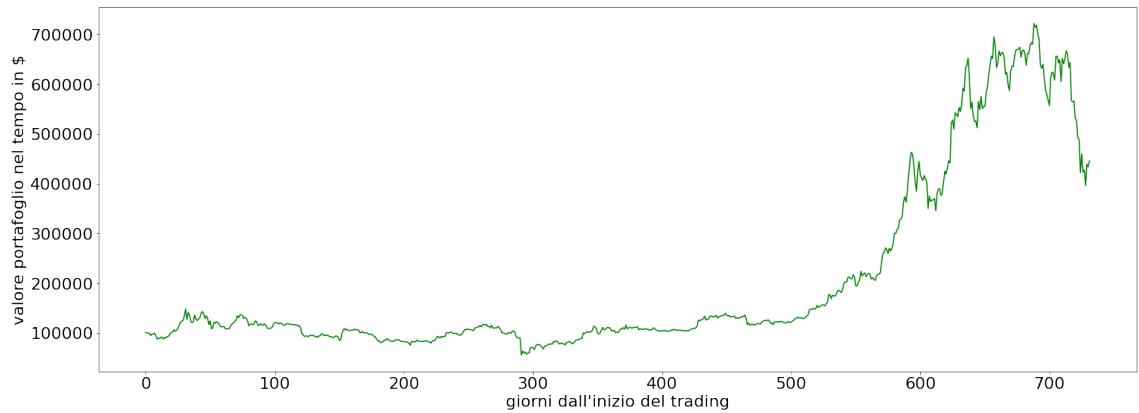


Figura 6.8: Performance portafoglio in 2 anni, con strategia buy and hold.

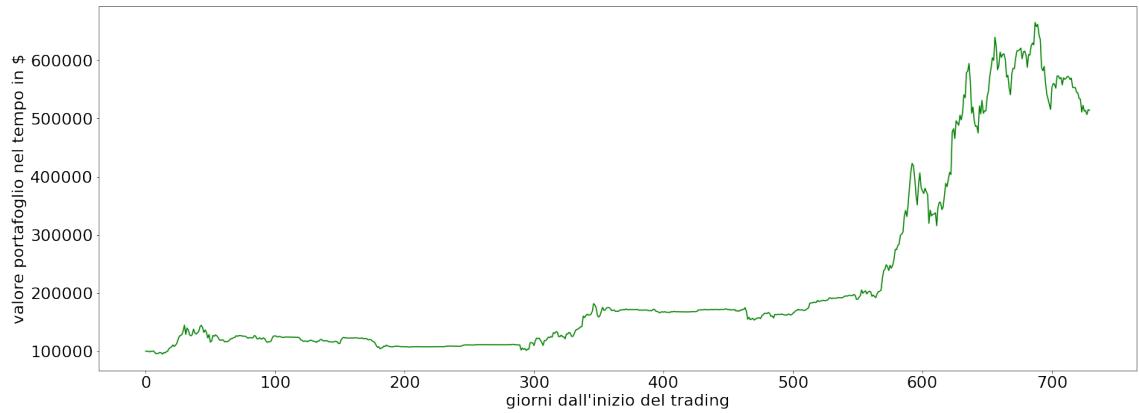


Figura 6.9: Performance portafoglio su 2 anni, utilizzando il modello random forest di classificazione.

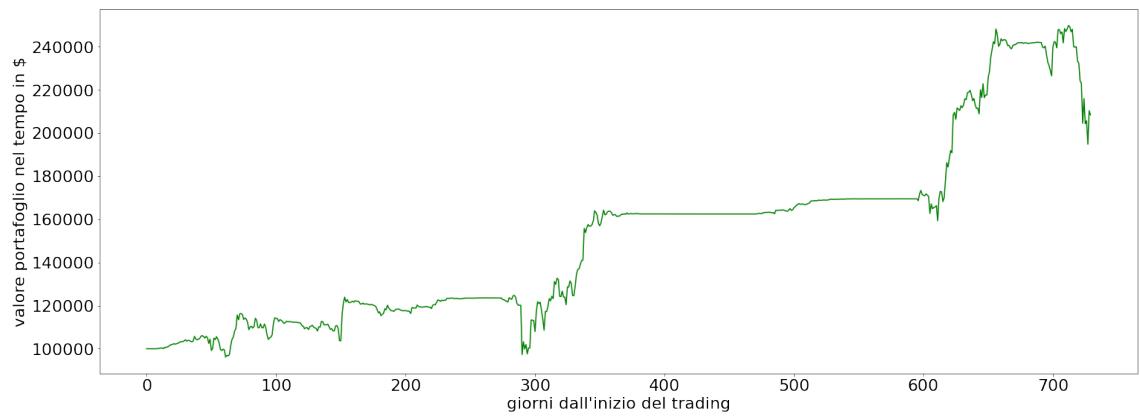


Figura 6.10: Performance portafoglio su 2 anni, utilizzando il modello random forest di regressione.

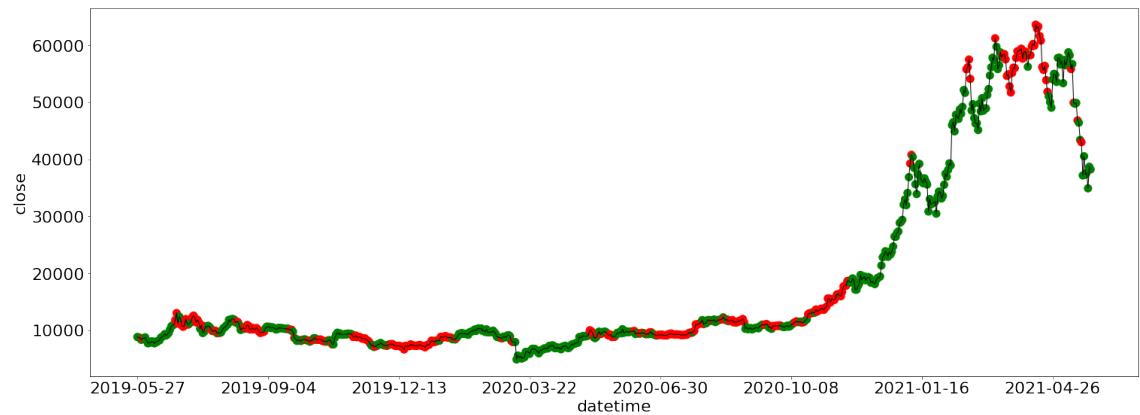


Figura 6.11: Previsioni del modello di classificazione random forest su 2 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

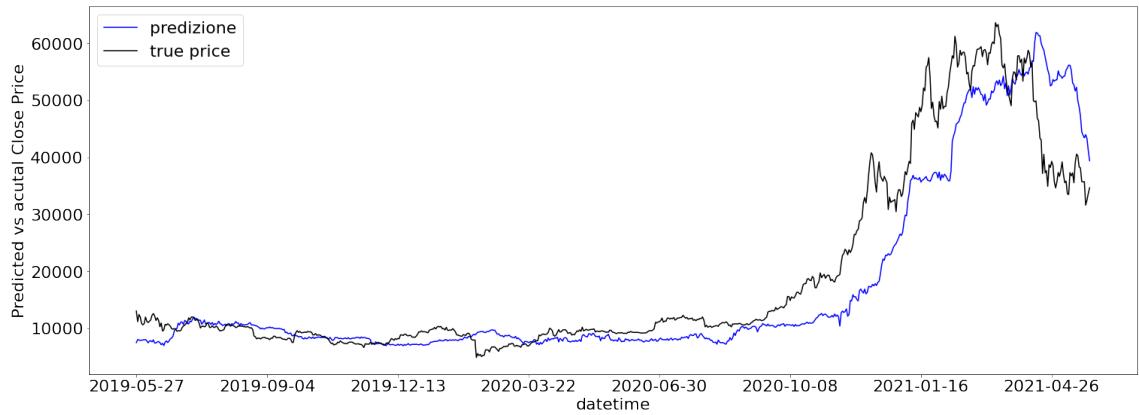


Figura 6.12: Previsioni del modello di regressione random forest su 2 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	68%	62%	65%	428
<b>1 (hold)</b>	0%	0%	0%	4
<b>2 (sell)</b>	52%	59%	55%	298
<b>accuracy del modello: 60.41%</b>				

Tabella 6.22: Performance del modello classificatore random forest su 2 anni di test.

<b>PERFORMANCE</b>	
<b>Mean Absolute Error (MAE)</b>	4307.711714981244
<b>Mean Squared Error (MSE)</b>	44638758.444507316
<b>Root Mean Squared Error (RMSE)</b>	6681.2243222711295
<b>Max Error</b>	23544.547547573115
<b>R<sup>2</sup></b>	0.8334481710168266

Tabella 6.23: Performance del modello regressore random forest su 2 anni di test.

### SIMULAZIONE 3 ANNI:

- **inizio:** 2018-05-27;
- **fine:** 2021-05-26;
- **prezzo iniziale BTC** 7364.09\$;
- **prezzo finale BTC** 39266.35\$.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
<b>strategia buy and hold</b>	532680.73\$	+432.68%
<b>classificazione utilizzando il dataset completo</b>	697966.45\$	+597,96%
<b>regressione utilizzando il dataset completo</b>	279921.24\$	+179,92%

Tabella 6.24: Comparazione performance su simulazione ultimi 3 anni.

In Figura 6.13, Figura 6.14 e Figura 6.15, sono riportate le prestazioni dei portafogli d’investimento utilizzando le diverse strategie. In Figura 6.16 e Figura 6.17, sono riportate graficamente le previsioni dei modelli. Come per la simulazione sugli ultimi 2 anni, anche questo caso, il classificatore riesce a battere le performance di una strategia **buy and hold** (con una differenza di profitto di oltre il 100%). In Tabella 6.25 e Tabella 6.26, sono riportate le metriche di valutazione per i diversi modelli di apprendimento automatico.

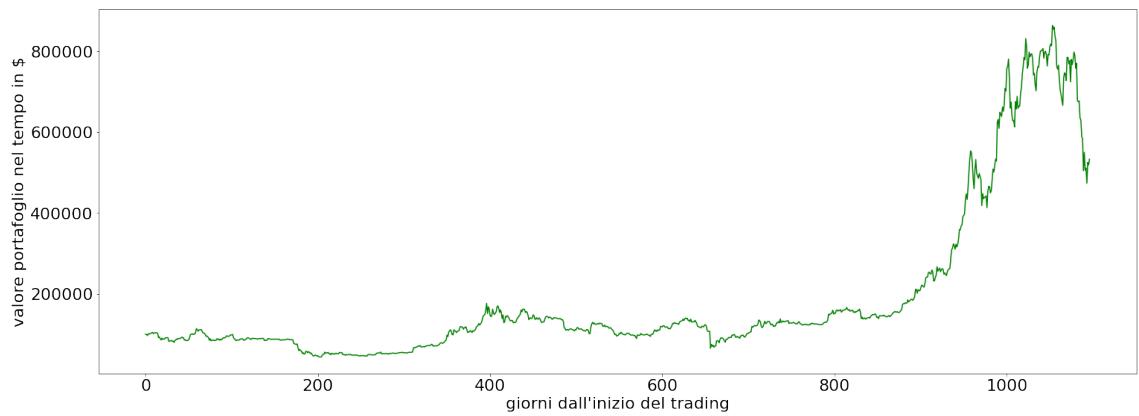


Figura 6.13: Performance portafoglio su 3 anni, con strategia buy and hold.

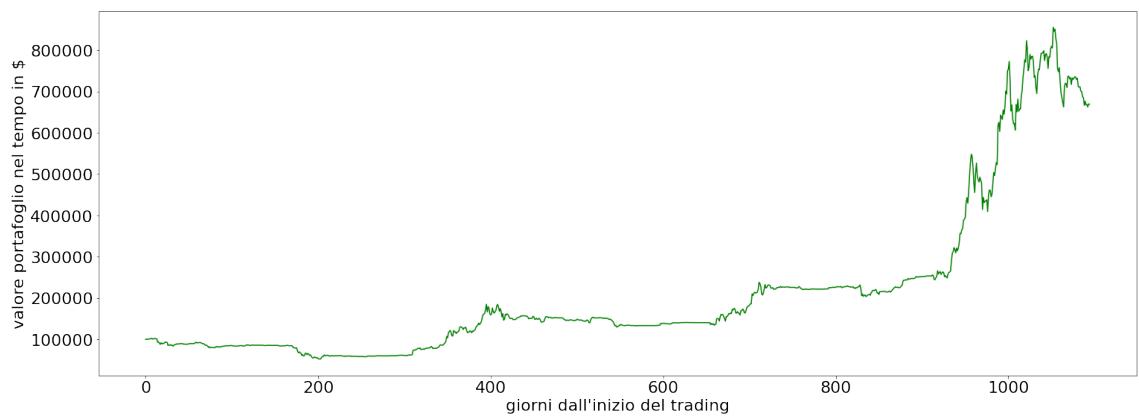


Figura 6.14: Performance portafoglio su 3 anni, utilizzando il modello random forest di classificazione.

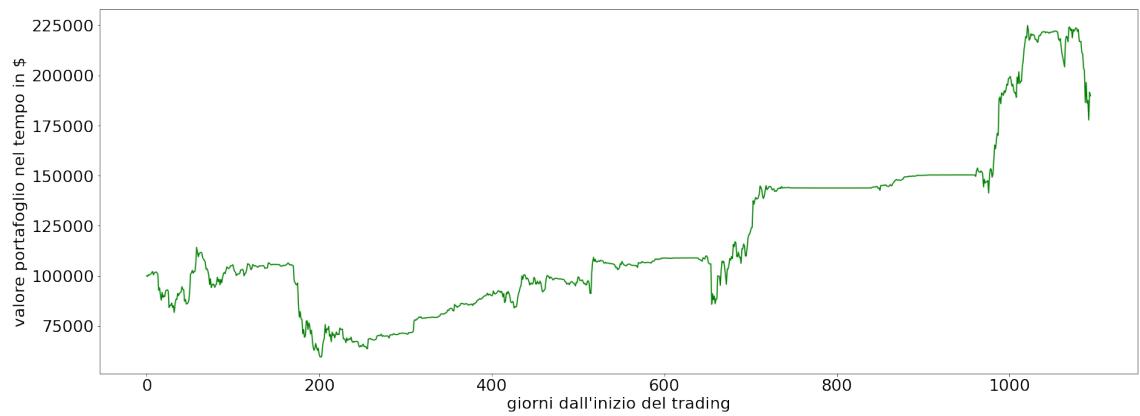


Figura 6.15: Performance portafoglio su 3 anni, utilizzando il modello random forest di regressione.

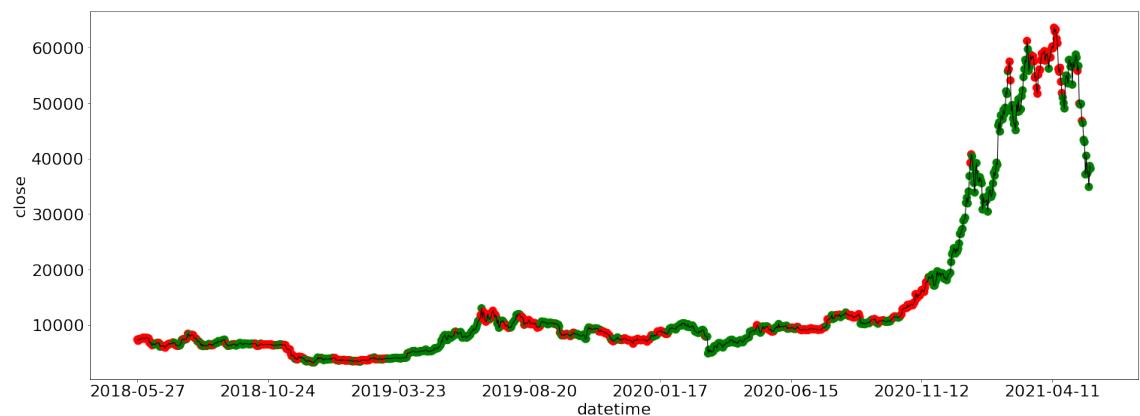


Figura 6.16: Previsioni del modello di classificazione random forest su 3 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

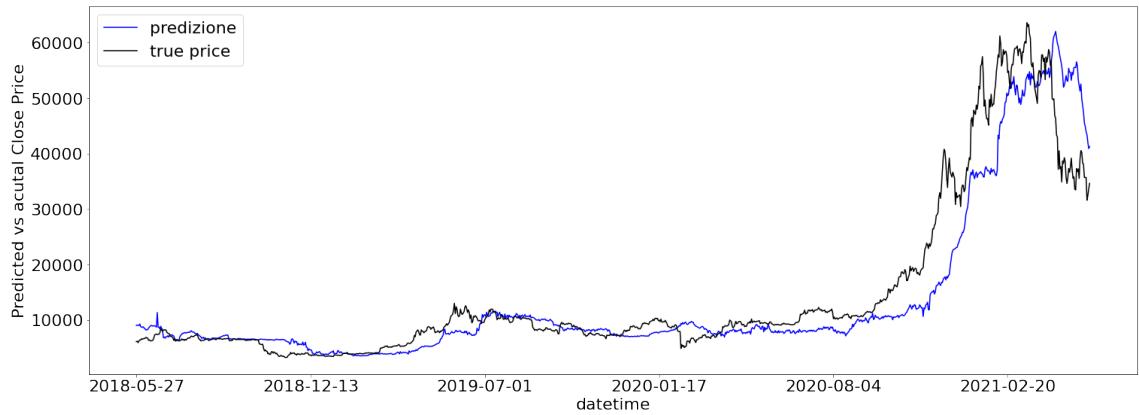


Figura 6.17: Previsioni del modello di regressione random forest su 3 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	66%	63%	64%	627
<b>1 (hold)</b>	1%	0%	0%	5
<b>2 (sell)</b>	53%	57%	55%	463
<b>accuracy del modello: 59.90%</b>				

Tabella 6.25: Performance del modello classificatore random forest su 3 anni di test.

PERFORMANCE	
<b>Mean Absolute Error (MAE)</b>	3260.775768907427
<b>Mean Squared Error (MSE)</b>	30719695.09938861
<b>Root Mean Squared Error (RMSE)</b>	5542.535078769336
<b>Max Error</b>	23449.73631783128
<b>R<sup>2</sup></b>	0.8599735570066722

Tabella 6.26: Performance del modello regressore random forest su 3 anni di test.

Risulta evidente come il modello di classificazione riesce a sovrapreformare il regressore. Per questo motivo nelle sezioni successive (nei modelli SVM e LSTM), dopo aver confermato nuovamente quest'affermazione (vedi Sezione 6.6.2), saranno valutate le sole performance dei modelli di classificazione. Non è necessario conoscere il prezzo reale di mercato per dover operare, nell'atto pratico, è sufficiente basarsi sull'aumentare o diminuire del prezzo futuro per effettuare azioni di compravendita. Ciò semplifica notevolmente il compito dei modelli di classificazione rispetto alla regressione, riflettendosi in prestazioni migliori. L'errore dei modelli regressori è ancor di più evidente dai grafici in Figura 6.7, Figura 6.12 e Figura 6.17. Si può notare come i grafici delle previsioni seguano quasi esattamente l'andamento del prezzo reale con un ritardo di circa 30 giorni (che rappresenta il time frame di previsione selezionato). Si può affermare che il modello basa la previsione del prezzo futuro sull'ultimo prezzo visto in allenamento, corrispondente a 30 giorni dal prezzo di previsione. Ciò rende il modello inaffidabile, nonostante in alcuni casi i test effettuati riportino un profitto (tali prestazioni possono essere giustificate da macro trend lunghi più di 30 giorni, come nella *bull run* di bitcoin a fine 2020, prolungata fino a Maggio 2021). Ovviamente, l'effetto di ritardo potrebbero verificarsi anche nel modello di classificazione (nonostante non risulti visibile graficamente). Tuttavia, dal momento che le prestazioni di quest'ultimo riescono a sovrapreformare il mercato, risulta ovvio che mediante la classificazione vengono apprese caratteristiche intrinseche all'andamento del prezzo. Nelle sezioni successive sono descritti i modelli SVM e LSTM, e nel Capitolo 7 è presentata un'analisi dettagliata dei risultati complessivi ottenuti.

## 6.6 Support Vector Machine

In questa sezione, sono riportati i risultati dei test ottenuti utilizzando modelli di apprendimento automatico SVM. Nella Sezione 6.6.1 viene determinato (come per il modello random forest) il valore  $n$  che garantisce performance migliori. Allo stesso modo delle simulazioni su random forest, sono testati entrambi i possibili dataset e ne sono confrontate le prestazioni. L'insieme di metriche che permette di ottenere risultati migliori è stato utilizzato per tutte le simulazioni successive. Nella Sezione 6.6.2 vi è un rapporto sul modello di regressione. Nella Sezione 6.6.3 sono analizzati i modelli su un intervallo di lungo termine (1 anno, 2 anni e 3 anni come per i modelli

random forest), confrontando le prestazioni con l’approccio *buy and hold*. In aggiunta, sono confrontate le performance a seconda dell’utilizzo di standardizzazione o normalizzazione come tecniche per il ridimensionamento dei dati (vedi Sezione 6.6.3). Per i test effettuati nella Sezione 6.6.1 e nella Sezione 6.6.2, i dati sono riportati in scala utilizzando la standardizzazione.

Per implementare il classificatore è stato utilizzato il modello *sklearn.svm.SVC*, *C-Support Vector Classification* [55]. Per il regressore è stato utilizzato il modello *sklearn.svm.SVR*, *Epsilon-Support Vector Regression* [55]. La configurazione è la seguente (gli altri parametri hanno i valori di default):

- **kernel:** *RBF*;
- **gamma:** *scale*;
- **C:** 1.0.

### 6.6.1 Classificatore

Per determinare il miglior valore  $n$ , è stato seguito lo schema di simulazione riportato nella Sezione 6.4 (come per i modelli random forest non sono stati tenuti in considerazione  $n > 30$  per il degradare delle performance). Nella Tabella 6.27 sono riportati i risultati ottenuti utilizzando il dataset ricavato dalla feature selection in Sezione 6.2.4.

Come per i modelli di apprendimento Random Forest, il valore  $n$  per cui si ottengono le migliori prestazioni è 30. Mediante la Tabella 6.28 e Tabella 6.29 è possibile notare come in questo caso utilizzando tutte le metriche a disposizione le performance vadano a peggiorare. Tutti i successivi test riportati sono quindi effettuati utilizzando il dataset ottenuto mediante feature selection nella Sezione 6.2.1. La GPU utilizzata in tutte e tre le simulazioni è: *NVIDIA-SMI 470.74 Driver Version: 460.32.03 CUDA Version: 11.2*. I tempi di esecuzione sono: circa 3 ore (simulazione generale con possibili valori  $n$ ), circa 40 minuti (simulazione  $n = 30$  con feature selezionate) e circa 60 minuti (simulazione  $n = 30$  con tutte le feature).

$n = 1$			
intervallo (in giorni)	test	accuracy	profitto
30	10	41.33%	-1.66%
90	10	49.11%	+20.32%
180	3	51.29%	+187.72%
365	1	45.47%	+15.36%
730	1	48.90%	+494.65%
$n = 7$			
intervallo (in giorni)	test	accuracy	profitto
30	10	52.66%	+6.02%
90	10	56.77%	+36.89%
180	3	50.92%	+18.91%
365	1	57.53%	+451.98%
730	1	53.69%	+519.74%
$n = 14$			
intervallo (in giorni)	test	accuracy	profitto
30	10	55.00%	+4.10%
90	10	56.22%	+23.15%
180	3	58.71%	+103.06%
365	1	49.31%	+21.86%
730	1	55.89%	+231.80%
$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	10	76.66%	+3.95%
90	10	59.00%	+27.79%
180	3	59.07%	+26.69%
365	1	57.53%	+20.43%
730	1	59.45%	+472.23%

Tabella 6.27: Risultati simulazione multi-periodo su modello classificatore SVM con l'obiettivo di determinare il miglior valore di  $n$ .

$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	20	62.50%	+4.66%
90	20	60.33%	+15.54%
180	6	63.42%	+72.19%
365	2	57.26%	+56.84%
730	2	59.45%	+427.49%

Tabella 6.28: Risultati simulazione multi-periodo su modello classificatore SVM prevedendo l’andamento del prezzo a 30 giorni nel futuro, utilizzando il dataset con feature selezionate in Sezione 6.2.4.

$n = 30$			
intervallo (in giorni)	test	accuracy	profitto
30	20	61.66%	+5.23%
90	20	57.94%	+25.69%
180	6	52.68%	+44.37%
365	2	58.08%	+66.85%
730	2	58.63%	+546.82%

Tabella 6.29: Risultati simulazione multiperiodo su modello classificatore SVM prevedendo l’andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili.

I risultati dimostrano che, a differenza dei modelli random forest, con il dataset ricavato in Sezione 6.2.4 si hanno prestazioni migliori (oltre ad un leggero aumento in accuratezza si ha anche un guadagno in tempi di esecuzione). Quest’ultimo è stato utilizzato per tutti i successivi test dei modelli SVM.

## 6.6.2 Regressore

Nella Tabella 6.30 sono riportati i risultati del modello SVM per il problema di regressione, utilizzando  $n = 30$ . Non è stata utilizzata la GPU fornita da Google Colab e il tempo di esecuzione richiesto è stato di 40 minuti circa. Il profitto ottenuto dal modello è positivo, nonostante ciò, le prestazioni sono inferiori al classificatore. Per questo motivo nella Sezione 6.6.3, in cui sono analizzate le performance su lunghi intervalli di tempo, non sono stati analizzati i risultati del modello SVM applicato al problema di regressione.

$n = 30$		
intervallo (in giorni)	test	profitto
30	20	+4.69%
90	20	+10.43%
180	6	+34.41%
365	2	+64.00%
730	2	+135.58%

Tabella 6.30: Risultati simulazione multiperiodo su modello regressore SVM prevedendo l'andamento del prezzo a 30 giorni nel futuro.

### 6.6.3 Analisi dei modelli su time frame ampio

In questa sezione sono riportati i risultati ottenuti dalle simulazioni dei modelli SVM negli ultimi anni, confrontandoli alla strategia *buy and hold* (per prestazioni strategia *buy and hold* vedi Sezione 6.5.3). Il classificatore è stato provato su ogni periodo utilizzando sia la standardizzazione che la normalizzazione (per ridimensionare i dati). Per ogni intervallo sono descritte le performance di entrambe le metodologie, ma i grafici esplicativi sono riportati solo per la migliore.

#### SIMULAZIONE 1 ANNO.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
strategia buy and hold	444247.47\$	+344.24%
classificazione utilizzando standardizzazione	350743.92\$	+250.74%
classificazione utilizzando normalizzazione	327157.72\$	+227.15%

Tabella 6.31: Comparazione performance su simulazione ultimo anno.

In Tabella 6.18 sono riportate le performance del portafoglio d'investimento utilizzando il modello SVM classificatore in cui i dati sono stati ridimensionati utilizzando la tecnica di standardizzazione. In Tabella 6.19 sono riportate graficamente le previsioni del modello. In Tabella 6.32 sono riportate le metriche di valutazione per il modello.

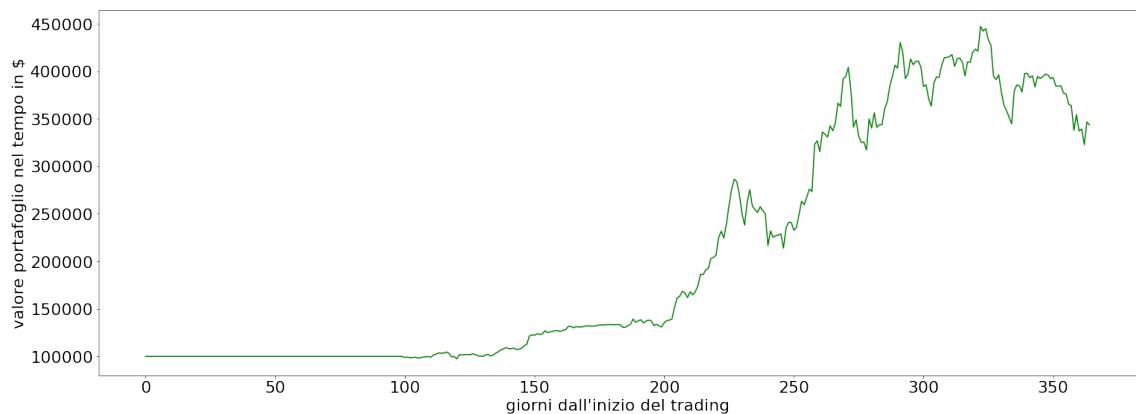


Figura 6.18: Performance portafoglio su un anno, utilizzando il modello SVM di classificazione (utilizzando standardizzazione).

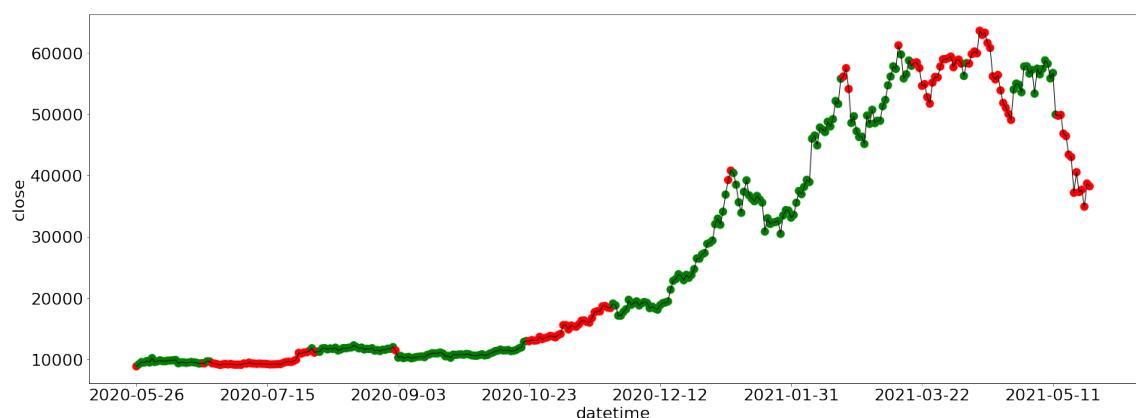


Figura 6.19: Previsioni del modello di classificazione SVM su un anno. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	74%	69%	71%	235
<b>1 (hold)</b>	100%	0%	0%	2
<b>2 (sell)</b>	50%	58%	54%	128
<b>accuracy del modello: 64.38%</b>				

Tabella 6.32: Performance del modello classificatore SVM su un anno di test (utilizzando standardizzazione).

### SIMULAZIONE 2 ANNI.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
<b>strategia buy and hold</b>	445622.70\$	+345.62%
<b>classificazione utilizzando standardizzazione</b>	401932.35\$	+301.93%
<b>classificazione utilizzando normalizzazione</b>	415746.26\$	+315.74%

Tabella 6.33: Comparazione performance su simulazione ultimi 2 anni.

Nella simulazione sugli ultimi 2 anni le performance migliori sono ottenuti utilizzando la tecnica di normalizzazione per il ridimensionamento dei dati (a differenza del test eseguito sull'ultimo anno, in cui si hanno performance migliori con la standardizzazione). In Figura 6.20 sono riportate le prestazioni dei portafogli d'investimento ottenute dal modello SVM di classificazione. In Figura 6.21 sono riportate graficamente le previsioni del modello. A differenza dei modelli basati su foreste casuali, nell'arco temporale di due anni, il classificatore SVM non riesce a battere le performance di una strategia **buy and hold**. In Tabella 6.34 sono riportate le metriche di valutazione per il modello di apprendimento automatico.

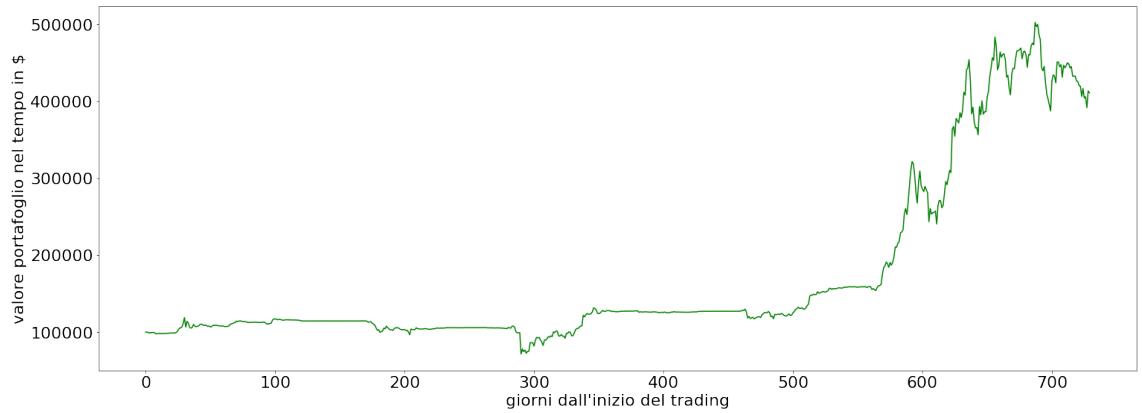


Figura 6.20: Performance portafoglio su 2 anni, utilizzando il modello SVM di classificazione (utilizzando normalizzazione).

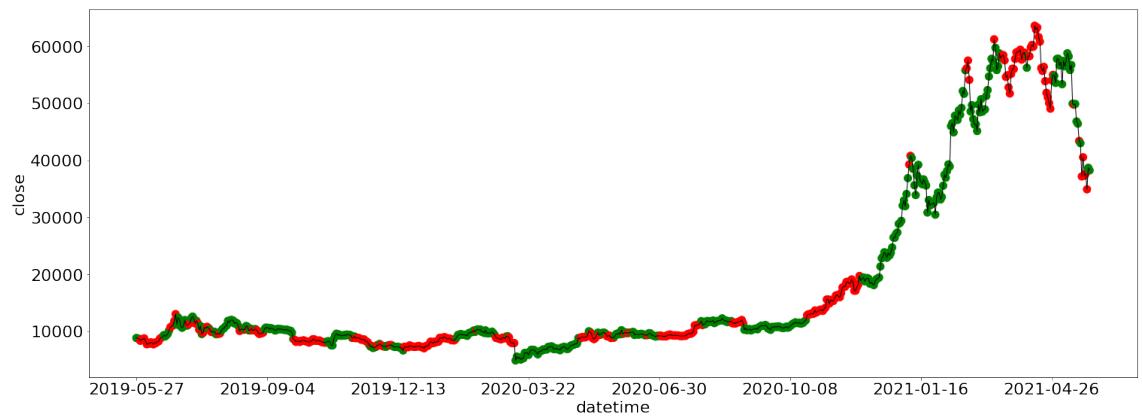


Figura 6.21: Previsioni del modello di classificazione SVM su 2 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	68%	54%	61%	428
<b>1 (hold)</b>	100%	0%	0%	4
<b>2 (sell)</b>	49%	63%	55%	298
<b>accuracy del modello: 57.67%</b>				

Tabella 6.34: Performance del modello classificatore SVM su 2 anni di test (utilizzando normalizzazione).

### SIMULAZIONE 3 ANNI.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
strategia buy and hold	532680.73\$	+432.68%
classificazione utilizzando standardizzazione	522753.09\$	+422.75%
classificazione utilizzando normalizzazione	548329.42\$	+448.32%
regressione utilizzando standardizzazione	232471.93\$	+132.47%

Tabella 6.35: Comparazione performance su simulazione ultimi 3 anni.

Nella simulazione sugli ultimi 3 anni le performance migliori sono ottenuti utilizzando la tecnica di normalizzazione per il ridimensionamento dei dati (come per il test sugli ultimi 2 anni), registrando un profitto superiore alla strategia *buy and hold*. In Tabella 6.6.3 sono riportate anche le performance ottenute utilizzando il modello SVM di regressione (allenato con il dataset in cui i dati sono stati riportati in scala con la tecnica di standardizzazione, in quanto con tale tecnica si ottengono risultati migliori). In Figura 6.22 e Figura 6.23 sono riportate le prestazioni dei portafogli d’investimento ottenute dai modelli SVM. In Figura 6.24 e Figura 6.25 sono riportate graficamente le previsioni dei modelli. In Tabella 6.36 e Tabella 6.37 sono riportate le metriche di valutazione per il modello di apprendimento automatico.



Figura 6.22: Performance portafoglio su 3 anni, utilizzando il modello SVM di classificazione (utilizzando normalizzazione).

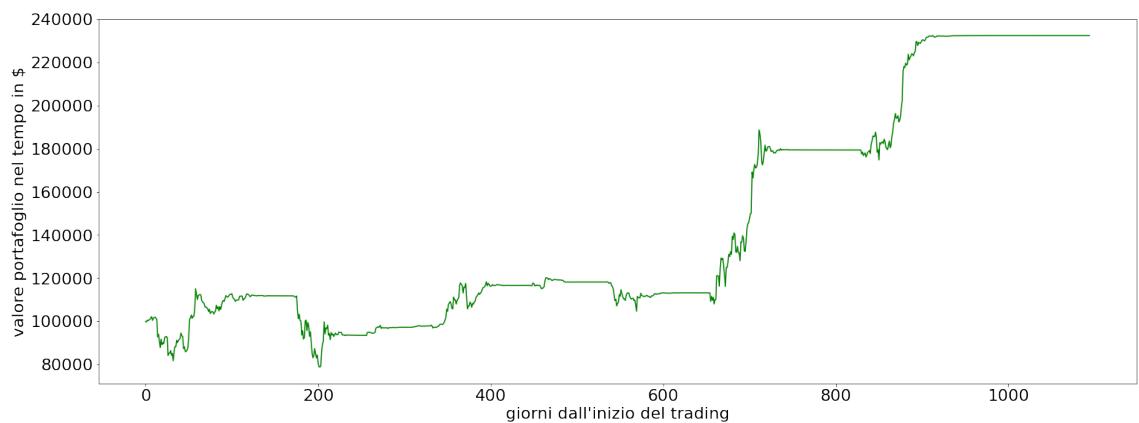


Figura 6.23: Performance portafoglio su 3 anni, utilizzando il modello random forest di regressione.

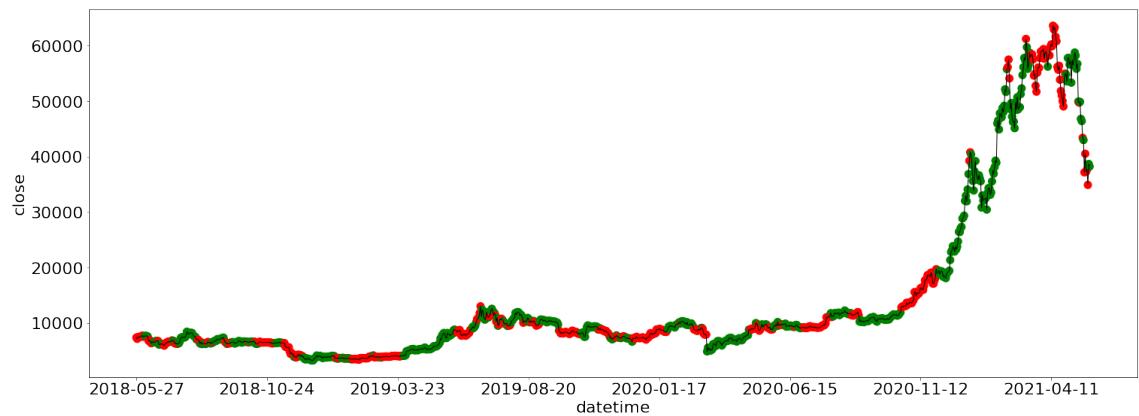


Figura 6.24: Previsioni del modello di classificazione SVM su 3 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.

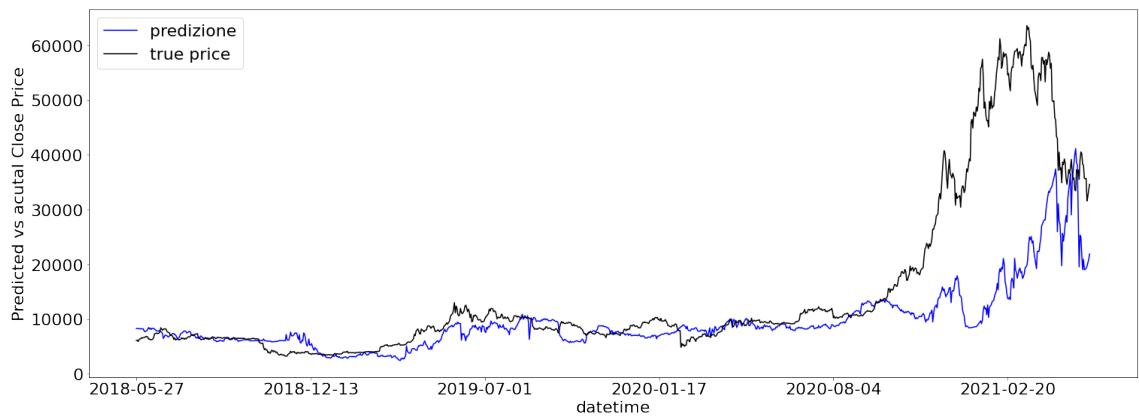


Figura 6.25: Previsioni del modello di regressione SVM su 3 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato (utilizzando standardizzazione).

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
<b>0 (buy)</b>	66%	57%	61%	627
<b>1 (hold)</b>	1%	0%	0%	5
<b>2 (sell)</b>	51%	61%	55%	463
<b>accuracy del modello: 58.08%</b>				

Tabella 6.36: Performance del modello classificatore SVM su 3 anni di test (utilizzando normalizzazione).

PERFORMANCE	
<b>Mean Absolute Error (MAE)</b>	5848.944680089
<b>Mean Squared Error (MSE)</b>	145921573.73671314
<b>Root Mean Squared Error (RMSE)</b>	12079.800235795008
<b>Max Error</b>	47632.50045385016
<b>R<sup>2</sup></b>	0.3348606208416697

Tabella 6.37: Performance del modello regressore SVM su 3 anni di test (utilizzando standardizzazione).

I risultati ottenuti confermano che battere il mercato rappresenta una sfida piuttosto ardua. In un time frame ampio, il modello di classificazione SVM riesce a ottenere un profitto superiore alla strategia *buy and hold* solo nei 3 anni. Per quanto riguarda il problema di *lag*, nei modelli di regressione (previsioni che dipendono in gran parte dall'ultimo dato visto in allenamento), nel caso delle SVM il problema sembra essere meno evidente. Nonostante ciò, l'errore di previsione del regressore è maggiore rispetto al modello basato su random forest (vedi Figura 6.25). Si può notare come il prezzo predetto non si avvicina mai ai massimi raggiunti realmente (questo causa un notevole calo del profitto). Le prestazioni del modello SVM risultano complessivamente leggermente inferiori rispetto ai modelli random forest. Tuttavia, i tempi di simulazione sono notevolmente ridotti nel caso di macchine a vettori di supporto. Nella Sezione 7 è descritta un'analisi accurata delle performance dei modelli.

## 6.7 Long Short-Term Memory

La definizione dello schema di una rete neurale, il suo allenamento e il successivo test, rappresentano i problemi di maggior difficoltà (sia a livello computazionale che implementativo) nell'approccio al Machine Learning. Per semplificare il lavoro necessario, sulla base degli studi precedenti e di primi test, si è scelto di implementare il solo modello di classificazione LSTM. Fin dai primi test, è stato possibile determinare che anche in questo modello le prestazioni migliori sono ottenute con un valore di  $n = 30$  (previsione a 30 giorni). Il modello di regressione, proprio come per Randomn Forest e SVM, garantisce prestazioni peggiori. Evitando di approfondire altre tipologie di modelli LSTM, come i regressori, è stato possibile incentrare maggiormente il lavoro sull'ottimizzazione del classificatore.

### 6.7.1 Implementazione e test

Nella fase di costruzione di una rete neurale, in particolare per reti LSTM, i principali parametri da definire sono:

- **Schema della rete neurale:** livelli, neuroni, funzioni di attivazione.
- **Numero di epochhe** per l'allenamento: un'epoca corrisponde al processamento di tutto il training set una volta.
- **Batch size:** in ogni epoca ci sono una o più iterazioni in cui vengono processati i dati di train. Questo perchè il dataset di allenamento non sempre è passato alla rete interamente, per problemi di esaurimento della memoria disponibile e di ottimizzazione. Si necessita quindi di suddividere il training set in sottoinsiemi da passare consecutivamente. Il batch size corrisponde al numero di record elaborati ad ogni sotto-iterazione (dimensione di ogni sottoinsieme).
- **History points** per il livello di input (vedi Sezione 6.3.2).

Nella fase iniziale, è stato implementato un modello di base per consentire la definizione dei primi iperparametri. Lo schema della rete è costituito da 5 livelli LSTM da 50 neuroni ciascuno e un livello di output per le tre classi con funzione di attivazione *softmax*.

Il primo parametro determinato è stato il numero di epoche necessarie all’allenamento per non occorrere in overfitting. Sono stati testati valori per 10, 50, 100, 200, 300, 500 e 1000 epoche. Analizzando le curve di apprendimento (vedi Figura 6.26, Figura 6.27 e Figura 6.28) e le performance sui set di validazione, è stato appurato che il range migliore per l’allenamento si aggira sulle 200 epoche.

In seguito, si è passati allo studio per definire il migliore valore di batch size. I valori testati sono: 1, 7, 32 e 64 (non si è andati oltre 64 in quanto le performance iniziano a degradare velocemente). I risultati ottenuti con un batch size di 7 risultano migliori sia a livello di performance che tempo di computazione richiesto.

Per scegliere il miglior valore per history points sono stati testati i valori 1, 5, 7, 30. Per la particolare struttura di rete, le migliori performance sono ottenute con un solo punto storico (vedi Figura 6.26 e Figura 6.26).

Si è passati infine all’ottimizzazione della rete neurale stessa, semplificando o aggiungendo complessità alla struttura. Sono stati aggiunti ulteriori livelli LSTM e anche livelli *Dense*, non riscontrando miglioramenti in prestazioni. Al contrario, semplificando il modello (rimuovendo livelli LSTM nascosti) le performance subiscono un leggero miglioramento (sia in accuratezza di previsione, che in tempo di computazione). La struttura finale della rete neurale è la seguente:

- **livello LSTM di input** da 51 neuroni (un neurone per ogni feature);
- **3 livelli nascosti LSTM** da 50 neuroni;
- **livello di output** con funzione di attivazione *softmax* per le 3 classi;
- **dropout** dopo ogni livello nascosto;
- **epoche di allenamento** = 200;
- **batch size** = 7;
- **history points** = 1;

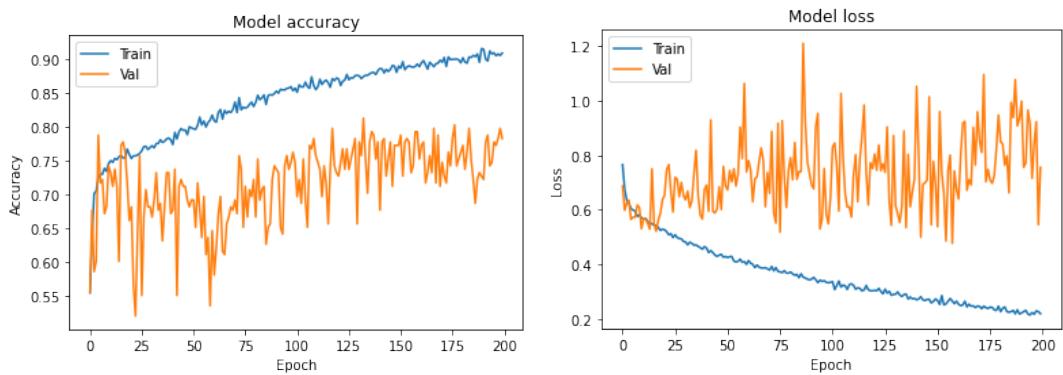


Figura 6.26: Curve di apprendimento utilizzando 200 epoche e history points = 1. Nell’immagine a sinistra è rappresentata l’accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell’immagine a destra si ha l’errore di previsione.

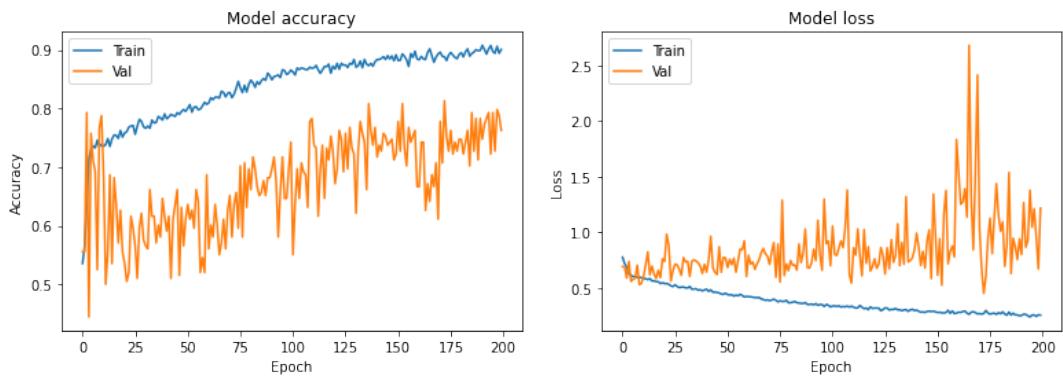


Figura 6.27: Curve di apprendimento utilizzando 200 epoche e history points = 7. Nell’immagine a sinistra è rappresentata l’accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell’immagine a destra si ha l’errore di previsione.

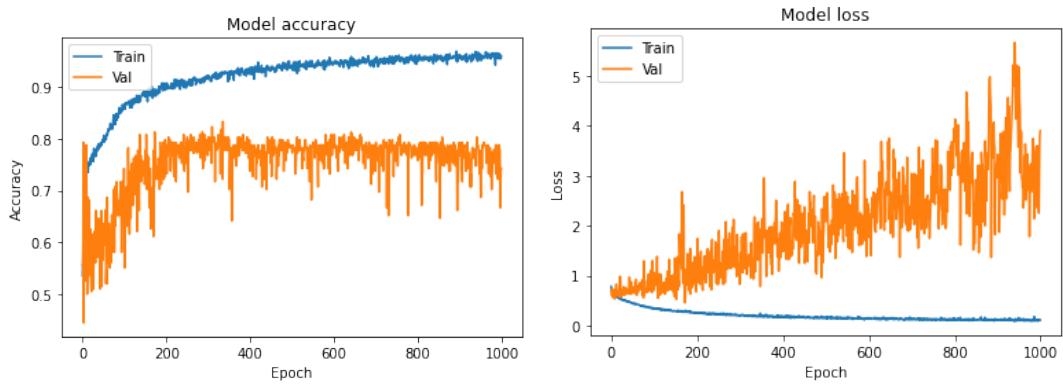


Figura 6.28: Curve di apprendimento utilizzando 1000 epoche e history points = 1. Nell’immagine a sinistra è rappresentata l’accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell’immagine a destra si ha l’errore di previsione.

Le performance migliori sono ottenute ridimensionando i dati utilizzando la normalizzazione. Per allenare e testare il modello è stata utilizzata la GPU: NVIDIA-SMI 470.74 Driver Version: 460.32.03 CUDA Version: 11.2, fornita da Colab. Nonostante ciò, il tempo necessario all’allenamento e al successivo test del modello, è notevolmente maggiore rispetto ai modelli SVM e Random Forest. Ad esempio, per effettuare una singola previsione, la fase di allenamento richiede circa 10 minuti. Dal momento che per ogni simulazione è richiesto di effettuare un numero di previsioni che va da 365 (simulazione su ultimo anno) a oltre 3000 (simulazione multi-periodo), il tempo necessario al test del modello utilizzando le tecniche in Sezione 6.4 risulta insostenibile. L’abbonamento *Colab Pro+* permette tempi di esecuzione di massimo 24 ore consecutive e ciò non ha reso possibile effettuare la simulazione del modello neanche su un anno consecutivo. Per questo motivo si è deciso di analizzare inizialmente solo gli ultimi 365 giorni, dividendo l’intervallo in 4 periodi (da 90, 90, 90 e 95 giorni) testati singolarmente. Ogni simulazione sul sotto-intervallo ha necessitato di circa 14 ore di esecuzione. I risultati sono stati analizzati singolarmente e aggregati.

In Tabella 6.38 è possibile notare come le prestazioni del modello LSTM risultano essere mediamente migliori dei modelli SVM e Random Forest. Tuttavia nell’intervallo che va dal 2020-11-22 al 2021-02-19 si riscontra un valore anomalo di accuracy. In tale periodo il modello LSTM non riesce a prevedere l’inizio dell’uptrend di fine 2020. Non è riscontrata una performance negativa nel portafoglio (non vengono ef-

fettuati acquisti), tuttavia il modello perde una grande opportunità di profitto. Le prestazioni ottenute in questo intervallo, se aggregate insieme agli altri intervalli, abbassano notevolmente le prestazioni complessive del modello (nonostante sembra che il modello LSTM riesca mediamente a sovrapreformare gli altri). Per questo motivo, per sfruttare a pieno le potenzialità della rete neurale LSTM, in Sezione 6.7.2 è stata implementata una soluzione al problema.

da 2020-05-26 a 2020-08-23			
	LSTM	SVM	Random Forest
<b>accuracy</b>	65.56%	56.66%	41.11%
da 2020-08-24 a 2020-11-21			
	LSTM	SVM	Random Forest
<b>accuracy</b>	51.11%	54.44%	33.33%
da 2020-11-22 a 2021-02-19			
	LSTM	SVM	Random Forest
<b>accuracy</b>	1.11%	86.66 %	97.77%
da 2021-02-20 a 2021-05-25			
	LSTM	SVM	Random Forest
<b>accuracy</b>	69.47%	48.42%	56.84%

Tabella 6.38: Comparazione performance LSTM, SVM, Random Forest su ultimo anno.

## 6.7.2 Sistema di voting multi-modello

L'anomalia di prestazione riscontrata nel periodo di simulazione che va dal 2020-11-22 al 2021-02-19 potrebbe essere causata dalla forte volatilità presente, e da notizie fondamentali che il modello non ha avuto modo di elaborare. Il modello non è riuscito a prevedere il rapido aumentare del prezzo di fine 2020. Tuttavia, anche sbagliando, il modello non ottiene performance negative sul portafoglio (il portafoglio al termine del periodo vale 100000\$, registrando una variazione dello 0%). In media le performance del modello LSTM sembrano comunque essere migliori degli altri modelli. Per questo motivo, è stato implementato un sistema in grado di combinare le decisioni di tutti i modelli implementati, con l'obiettivo di ottenere le migliori performance possibili. Il

sistema implementato consiste nell'aggregare le previsioni dei 3 classificatori e scegliere come etichetta *finale* la maggioranza delle decisioni. Ad esempio, se due modelli restituiscono l'etichetta **buy** e un altro **sell**, la decisione finale sarà data dell'etichetta di maggioranza **buy** (nel caso di una situazione di parità, la scelta di maggioranza è quella del modello LSTM). I risultati dimostrano che questa tecnica permette di ottenere le prestazioni migliori, sovrapassando qualsiasi singolo modello in accuracy e profitto. In Tabella 6.39 e in Tabella 6.40 sono riportati i risultati e la comparazione con gli altri modelli. In Figura 6.29 e in Figura 6.30 sono riportate graficamente le previsioni del sistema di voting e il portafoglio nell'ultimo anno.

da 2020-05-26 a 2020-08-23				
	LSTM	SVM	Random Forest	Sistema di voting
<b>accuracy</b>	65.56%	56.66%	41.11%	53.33%
da 2020-08-24 a 2020-11-21				
	LSTM	SVM	Random Forest	Sistema di voting
<b>accuracy</b>	51.11%	54.44%	33.33%	52.22%
da 2020-11-22 a 2021-02-19				
	LSTM	SVM	Random Forest	Sistema di voting
<b>accuracy</b>	1.11%	86.66 %	97.77%	86.66%
da 2021-02-20 a 2021-05-25				
	LSTM	SVM	Random Forest	Sistema di voting
<b>accuracy</b>	69.47%	48.42%	56.84%	58.94%

Tabella 6.39: Comparazione performance LSTM, SVM, Random Forest e sistema di voting su ultimo anno (su singoli periodi).

<b>accuracy del sistema di voting: 66.57%</b>
<b>accuracy del modello SVM: 64.38%</b>
<b>accuracy del modello Random Forest: 59.17%</b>

Tabella 6.40: Comparazione performance SVM, Random Forest e sistema di voting su ultimo anno.

## SIMULAZIONE 1 ANNO.

PRESTAZIONI		
METODO	VALORE PORTAFOGLIO	PROFITTO
<b>strategia buy and hold</b>	444247.47\$	+344.24%
<b>Sistema di voting (LSTM, SVM, Random Forest)</b>	364255.13\$	+264.25%
<b>SVM</b>	350743.92\$	+250.74%
<b>Random Forest</b>	305100.00\$	+205,10%

Tabella 6.41: Comparazione performance su simulazione ultimo anno.

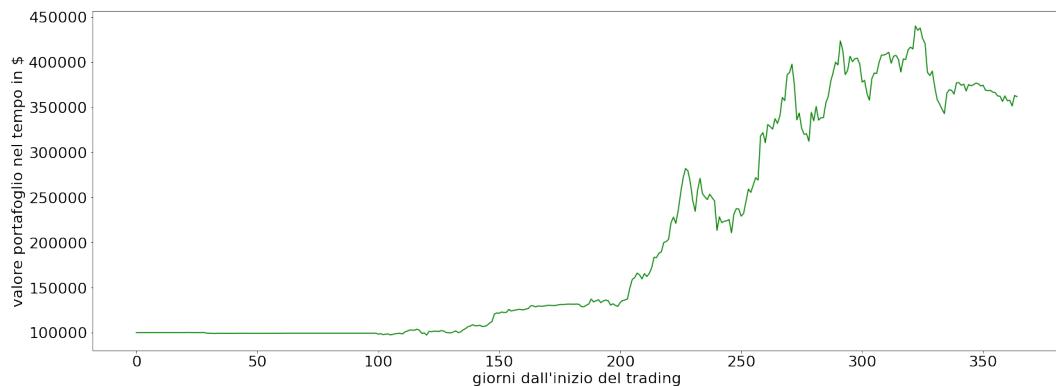


Figura 6.29: Performance portafoglio su 1 anno, utilizzando il sistema di voting.

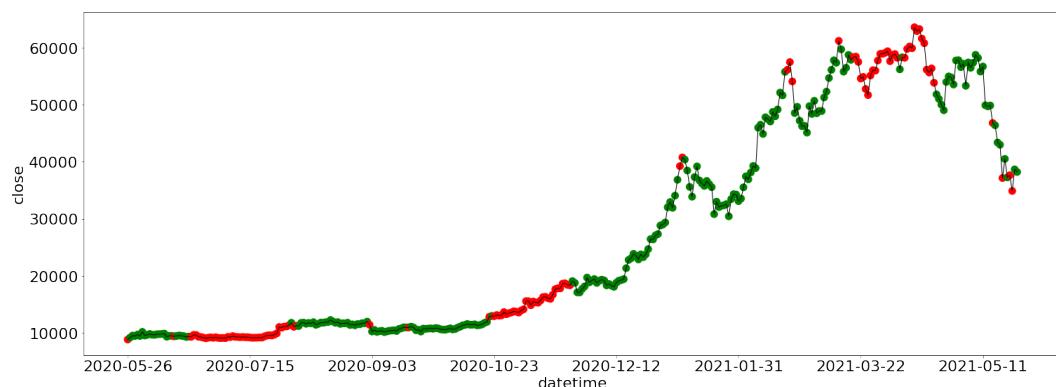


Figura 6.30: Previsioni del sistema di voting su ultimo anno.

# Capitolo 7

## Conclusioni

### 7.1 Analisi dei risultati

I risultati ottenuti dimostrano che ottenere un buon profitto, da un sistema di trading basato sulle previsioni di modelli di Machine Learning, è possibile. Nonostante la difficoltà del problema, i modelli di apprendimento automatico riescono ad apprendere caratteristiche intrinseche all'andamento del mercato, permettendone la previsione. Un elemento di rilevante importanza è costituito dai dati a disposizione. È infatti stato possibile raccogliere 51 metriche di analisi tecnica e fondamentale per Bitcoin. La parte di analisi e raccolta dati è di fondamentale importanza nel Machine Learning, per questo motivo la ricerca effettuata vi si è focalizzata molto. L'utilizzo combinato di analisi tecnica e analisi fondamentale permette di analizzare in modo accurato il mercato delle criptovalute. Si può inoltre dimostrare che anche in periodi di down-trend (discesa prolungata del prezzo di Bitcoin), le previsioni dei modelli riescono a discostarsi dall'andamento di mercato, permettendo di ottenere un profitto. Per evidenziare l'efficacia dei modelli, in Tabella 7.1 e in Tabella 7.2, sono riportati alcuni risultati dei molteplici test in grado di battere il mercato. Si può notare come in periodi di lateralizzazione o di forte discesa, le previsioni dei modelli permettono di ottenere performance positive sul portafoglio d'investimento, a differenza di quanto si otterrebbe utilizzando una strategia di *buy and hold*.

La tecnica di trading utilizzata nelle simulazioni è stata appositamente studiata per essere il più semplice possibile, in modo da far dipendere i profitti maggiormente dalle previsioni del modello (per evidenziarne l'efficacia di previsione). Si noti dal primo

esempio in Tabella 7.1, che anche con il 100% di accuratezza il modello non riesce a battere il mercato. Il motivo è che la strategia di trading utilizzata muove solo il 10% del portafoglio in questione per ogni azione. Nella Sezione 7.2 sono descritte possibili migliorie anche per il sistema di trading, con il fine di massimizzarne i profitti.

modello	accuracy	performance	info periodo	buy and hold
<b>Random Forest</b>	100%	+39.51%	da 2020-11-26 a 2020-12-26. Prezzo iniziale btc: 17106.29\$. Prezzo finale btc: 26440.24\$	+54.40%
<b>Random Forest</b>	93.33%	+9.66%	da 2020-03-02 a 2020-04-01. Prezzo iniziale btc: 8865.83\$. Prezzo finale btc: 6589.24\$	-25.75%
<b>Random Forest</b>	57.26%	+42.38%	da 2019-09-17 a 2020-09-16. Prezzo iniziale btc: 10241.27\$. Prezzo finale btc: 10974.90\$	+7.05%
<b>Random Forest</b>	71.11%	+18.28%	da 2019-12-26 a 2020-03-25. Prezzo iniziale btc: 7240.02\$. Prezzo finale btc: 6677.07\$	-7.86%

Tabella 7.1: Esempi di report su simulazioni modelli. Parte 1.

modello	accuracy	performance	info periodo	buy and hold
<b>SVM</b>	43.33%	-0.09%	da 2019-09-14 a 2019-10-14. Prezzo iniziale btc: 10358.04\$. Prezzo finale btc: 8371.98\$	-19.25%
<b>SVM</b>	96.66%	+3.45%	da 2020-08-18 a 2020-09-17. Prezzo iniziale btc: 11988.74\$. Prezzo finale btc: 10948.98\$	-8.76%
<b>SVM</b>	74.44%	+44.63%	da 2020-03-06 a 2020-06-04. Prezzo iniziale btc: 9120.74\$. Prezzo finale btc: 9803.70\$	+7.38%
<b>LSTM</b>	69.47%	+1.84%	da 2021-02-20 a 2021-05-26. Prezzo iniziale btc: 56134.54\$. Prezzo finale btc: 39266.35\$	-30.11%

Tabella 7.2: Esempi di report su simulazioni modelli. Parte 2.

## 7.2 Lavori futuri

Nella ricerca di questa tesi sono stati sviluppati e testati numerosi modelli di apprendimento automatico, nel tentativo di prevedere l'andamento di prezzo del Bitcoin. I risultati evidenziano che ottenere performance superiori a quelle di mercato, in una

strategia di investimento, risulta essere molto difficile. Tuttavia, è stato dimostrato che ciò è possibile, sia nel breve periodo che in un'ottica a lungo termine. Lo studio di questa tesi ha posto le basi a continui e numerosi lavori futuri in questo ambito. Numerose migliorie e accorgimenti sono possibili. È ovviamente possibile effettuare una più accurata ottimizzazione degli iperparametri (per i modelli). Sono implementabili anche migliorie ai modelli stessi, creando reti neurali più complesse, approfondendo le LSTM ma anche utilizzando strutture di rete diverse come CNN (convolutional neural network). In futuro sarà sicuramente possibile reperire altre metriche, tra le più importanti vi sono i movimenti legati agli enti istituzionali, ad aziende come Grayscale e MicroStrategy, e a fondi d'investimento come gli ETF (i cui dati saranno sicuramente analizzati e raccolti da numerosi analisti). Allo stesso modo si potrebbero testare le performance su un dataset orario (invece che giornaliero) e sviluppare specifiche strategie di trading intraday. Migliorie e ulteriori implementazioni potrebbero essere applicate anche alla strategia di trading, ad esempio utilizzando un sistema in grado di operare sia LONG che SHORT (vendita allo scoperto, per sfruttare i downtrend). Tra le altre migliorie possibili, si potrebbe cercare di ridurre al minimo il numero di azioni eseguite in compravendita (per non spendere troppo in commissioni), concentrando i volumi di movimento in poche operazioni ma più efficaci.

I risultati ottenuti al momento sono molto positivi, e forniscono la base per ulteriori lavori e studi. Grazie alle numerose simulazioni, anche in intervalli annuali, si può affermare che è possibile ottenere profitto da un sistema di trading basato sulle previsioni di modelli ad apprendimento automatico nel mercato di Bitcoin. Risulta possibile prevedere il trend di prezzo e sfruttare l'estrema volatilità. Nell'ottica di un investitore, le performance ottenute dimostrano che una simile strategia può rivelarsi utile ed efficiente, sia nel caso di investimenti a lungo che a breve termine. In futuro saranno quindi effettuati dei test in tempo reale, integrando il sistema in una piattaforma di trading (Exchange) (partendo da un portafoglio in demo fino ad arrivare, in caso di risultati positivi, ad un'applicazione reale).

# Elenco delle figure

2.1	Max supply di Bitcoin [8]. . . . .	11
2.2	Struttura di un blocco della Blockchain Bitcoin [4]. . . . .	12
2.3	Struttura dell' <i>header</i> di un blocco della Blockchain Bitcoin [4]. . . . .	13
2.4	Esempio di blockchain [4]. . . . .	14
3.1	Bitcoin: Hash Rate, periodo: Luglio 2015/Giugno 2021, intervallo giornaliero. . . . .	22
4.1	Artificial Intelligence, Machine Learning, Deep Learning . . . . .	24
4.2	Esempi di classificazione lineare ( <i>sinistra</i> ) e classificazione a più dimensioni ( <i>destra</i> ). . . . .	26
4.3	Esempio di regressione. . . . .	27
4.4	Esempio di albero decisionale binario con variabili discrete. . . . .	28
4.5	Esempio del funzionamento di un modello basato su foresta casuale per problemi di classificazione. . . . .	33
4.6	Esempio di separazione lineare di una SVM [64]. . . . .	35
4.7	Esempio di trasformazione di un problema non linearmente separabile in un problema linearmente separabile, mediante l'utilizzo di una funzione Kernel [65]. . . . .	39
4.8	SVR con diverse soglie $\varepsilon$ . All'aumentare di $\varepsilon$ , la previsione diventa meno sensibile agli errori [66]. . . . .	40
4.9	Esempio di funzionamento di un modello Perceptron [68]. . . . .	42
4.10	Esempio di una rete neurale artificiale multilayer feed-forward. . . . .	46
4.11	Esempio di superficie di errore con applicazione del metodo di discesa del gradiente [29] [30]. . . . .	48
4.12	Funzionamento interno di una cella RNN [47]. . . . .	50

4.13	Funzionamento interno di una cella LSTM [47]. . . . .	51
4.14	Rappresentazione del funzionamento dell'Input Gate [47]. . . . .	52
4.15	Rappresentazione del funzionamento del Forget Gate [47]. . . . .	52
4.16	Rappresentazione del funzionamento dell'Output Gate [47]. . . . .	53
4.17	Nell'immagine di sinistra, la linea verde rappresenta un modello con overfitting, mentre la linea nera rappresenta un modello con un buon tasso di generalizzazione. Nell'immagine di destra, la linea blu rappresenta un modello che ha appreso anche caratteristiche sui dati di rumore, portandolo ad un sovradattamento, mentre la linea nera rappresenta invece un modello molto più semplice in grado di generalizzare meglio [15, 16]. . . . .	56
4.18	Esempio di curve di apprendimento di una rete neurale utilizzando 14 epoche. Nell'immagine a sinistra è rappresentata l'accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell'immagine a destra si ha l'errore di previsione. . . . .	56
4.19	Matrice di confusione per problema di classificazione binaria dove A e B sono rispettivamente le due classi [72]. . . . .	57
6.1	Distribuzione binomiale e posizionamento delle feature [48]. . . . .	75
6.2	Esempio di matrice di correlazione. . . . .	77
6.3	Performance portafoglio su un anno con strategia buy and hold. . . .	102
6.4	Performance portafoglio su un anno, utilizzando il modello random forest di classificazione (dataset completo, ricavato con feature selection con $n = 30$ ). . . . .	102
6.5	Performance portafoglio su un anno, utilizzando il modello random forest di regressione. . . . .	103
6.6	Previsioni del modello di classificazione random forest su un anno, utilizzando il dataset completo. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta.	103

6.7	Previsioni del modello di regressione random forest su un anno, utilizzando il dataset completo. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato. . . . .	104
6.8	Performance portafoglio in 2 anni, con strategia buy and hold. . . . .	106
6.9	Performance portafoglio su 2 anni, utilizzando il modello random forest di classificazione. . . . .	106
6.10	Performance portafoglio su 2 anni, utilizzando il modello random forest di regressione. . . . .	107
6.11	Previsioni del modello di classificazione random forest su 2 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta. . . . .	107
6.12	Previsioni del modello di regressione random forest su 2 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato. . . . .	108
6.13	Performance portafoglio su 3 anni, con strategia buy and hold. . . . .	110
6.14	Performance portafoglio su 3 anni, utilizzando il modello random forest di classificazione. . . . .	110
6.15	Performance portafoglio su 3 anni, utilizzando il modello random forest di regressione. . . . .	111
6.16	Previsioni del modello di classificazione random forest su 3 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta. . . . .	111
6.17	Previsioni del modello di regressione random forest su 3 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato. . . . .	112
6.18	Performance portafoglio su un anno, utilizzando il modello SVM di classificazione (utilizzando standardizzazione). . . . .	118

6.19 Previsioni del modello di classificazione SVM su un anno. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta. . . . .	118
6.20 Performance portafoglio su 2 anni, utilizzando il modello SVM di classificazione (utilizzando normalizzazione). . . . .	120
6.21 Previsioni del modello di classificazione SVM su 2 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta. . . . .	120
6.22 Performance portafoglio su 3 anni, utilizzando il modello SVM di classificazione (utilizzando normalizzazione). . . . .	122
6.23 Performance portafoglio su 3 anni, utilizzando il modello random forest di regressione. . . . .	122
6.24 Previsioni del modello di classificazione SVM su 3 anni. Il grafico rappresenta l'andamento del mercato reale. Nelle aree di colore verde la previsione del modello è stata corretta. Nelle aree di colore rosso la previsione non è stata corretta. . . . .	123
6.25 Previsioni del modello di regressione SVM su 3 anni. La linea di colore blu rappresenta le previsioni del modello. La linea di colore nero rappresenta i prezzi reali di mercato (utilizzando standardizzazione). .	123
6.26 Curve di apprendimento utilizzando 200 epoche e history points = 1. Nell'immagine a sinistra è rappresentata l'accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell'immagine a destra si ha l'errore di previsione. . . . .	127
6.27 Curve di apprendimento utilizzando 200 epoche e history points = 7. Nell'immagine a sinistra è rappresentata l'accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell'immagine a destra si ha l'errore di previsione. . . . .	127

6.28 Curve di apprendimento utilizzando 1000 epoche e history points = 1.	Nell'immagine a sinistra è rappresentata l'accuratezza ottenuta dal modello sui dati di training e sul set di validazione con il passare delle epoche. Rispettivamente nell'immagine a destra si ha l'errore di previsione. . . . .	128
6.29 Performance portafoglio su 1 anno, utilizzando il sistema di voting. .	. . . . .	131
6.30 Previsioni del sistema di voting su ultimo anno. . . . .	. . . . .	131

# Elenco delle tabelle

5.1	Tabella di riepilogo ricerche correlate. . . . .	64
6.1	Descrizione Feature parte 1. . . . .	66
6.2	Descrizione Feature parte 2. . . . .	67
6.3	Descrizione Feature parte 3. . . . .	68
6.4	Descrizione Feature parte 4. . . . .	69
6.5	Descrizione Feature parte 5. . . . .	70
6.6	Risultati Boruta sul dataset con prezzo di chiusura traslato. . . . .	81
6.7	Risultati Boruta sul dataset con prezzo di chiusura non traslato. Parte 1. . . . .	82
6.8	Risultati Boruta sul dataset con prezzo di chiusura non traslato. Parte 2. . . . .	83
6.9	Risultati Pearson e Spearman sul dataset con prezzo di chiusura traslato. Parte 1. . . . .	83
6.10	Risultati Pearson e Spearman sul dataset con prezzo di chiusura traslato. Parte 2. . . . .	84
6.11	Risultati complessivi feature selection. . . . .	85
6.12	Tabella feature selezionate in ordine di importanza. . . . .	87
6.13	Risultati simulazione multi-periodo su modello classificatore Random Forest con l'obiettivo di determinare il miglior valore di $n$ . . . . .	97
6.14	Risultati simulazione multi-periodo su modello classificatore Random Forest prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando il dataset con feature selezionate in Sezione 6.2.4. . . . .	98
6.15	Risultati simulazione multi-periodo su modello classificatore Random Forest prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili. . . . .	98

6.16 Risultati simulazione multi-periodo su modello regressore Random Forest prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili. . . . .	99
6.17 Comparazione performance su simulazione ultimo anno. . . . .	101
6.18 Performance del modello classificatore random forest su un anno di test utilizzando il dataset ottenuto mediante feature selection in Sezione 6.2.4.104	
6.19 Performance del modello classificatore random forest su un anno di test utilizzando il dataset con tutte le metriche. . . . .	104
6.20 Performance del modello regressore random forest su un anno di test utilizzando il dataset con tutte le metriche. . . . .	105
6.21 Comparazione performance su simulazione ultimi 2 anni. . . . .	105
6.22 Performance del modello classificatore random forest su 2 anni di test.	108
6.23 Performance del modello regressore random forest su 2 anni di test. .	108
6.24 Comparazione performance su simulazione ultimi 3 anni. . . . .	109
6.25 Performance del modello classificatore random forest su 3 anni di test.	112
6.26 Performance del modello regressore random forest su 3 anni di test. .	112
6.27 Risultati simulazione multi-periodo su modello classificatore SVM con l'obiettivo di determinare il miglior valore di $n$ . . . . .	115
6.28 Risultati simulazione multi-periodo su modello classificatore SVM prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando il dataset con feature selezionate in Sezione 6.2.4. . . . .	116
6.29 Risultati simulazione multiperiodo su modello classificatore SVM prevedendo l'andamento del prezzo a 30 giorni nel futuro, utilizzando tutte le metriche disponibili. . . . .	116
6.30 Risultati simulazione multiperiodo su modello regressore SVM prevedendo l'andamento del prezzo a 30 giorni nel futuro. . . . .	117
6.31 Comparazione performance su simulazione ultimo anno. . . . .	117
6.32 Performance del modello classificatore SVM su un anno di test (utilizzando standardizzazione). . . . .	119
6.33 Comparazione performance su simulazione ultimi 2 anni. . . . .	119
6.34 Performance del modello classificatore SVM su 2 anni di test (utilizzando normalizzazione). . . . .	120
6.35 Comparazione performance su simulazione ultimi 3 anni. . . . .	121

6.36 Performance del modello classificatore SVM su 3 anni di test (utilizzando normalizzazione). . . . .	124
6.37 Performance del modello regressore SVM su 3 anni di test (utilizzando standardizzazione). . . . .	124
6.38 Comparazione performance LSTM, SVM, Random Forest su ultimo anno. . . . .	129
6.39 Comparazione performance LSTM, SVM, Random Forest e sistema di voting su ultimo anno (su singoli periodi). . . . .	130
6.40 Comparazione performance SVM, Random Forest e sistema di voting su ultimo anno. . . . .	130
6.41 Comparazione performance su simulazione ultimo anno. . . . .	131
7.1 Esempi di report su simulazioni modelli. Parte 1. . . . .	133
7.2 Esempi di report su simulazioni modelli. Parte 2. . . . .	134

# Bibliografia

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- [2] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.
- [3] Analisi fondamentale nelle criptovalute. Guida all’analisi fondamentale delle criptovalute, 09 2021. URL <https://academy.binance.com/it/articles/a-guide-to-cryptocurrency-fundamental-analysis>.
- [4] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.
- [5] Lauren Barghout. Spatial-taxon information granules as used in iterative fuzzy-decision-making for image segmentation. In *Granular computing and decision-making*, pages 285–318. Springer, 2015.

- [6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
- [7] Aniruddha Bhandari. Feature scaling for machine learning: Understanding the difference between normalization vs. standardization, 04 2020. URL <https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/>.
- [8] Bitcoin max supply. File:controlled supply-supply over block height.png, 09 2021. URL [https://en.bitcoin.it/wiki/File:Controlled\\_supply-supply\\_over\\_block\\_height.png](https://en.bitcoin.it/wiki/File:Controlled_supply-supply_over_block_height.png).
- [9] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992. URL <http://www.gautampendse.com/projects/bsvm/webpage/boser1992.pdf>.
- [10] Leo Breiman. Random forests. *Machine Learning*, 45, 2001. doi: 10.1023/A:1010933404324. URL <https://doi.org/10.1023/A:1010933404324>.
- [11] Jason Brownlee. How to reshape input data for long short-term memory networks in keras, 08 2017. URL <https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/>.
- [12] James Chen. Exponential moving average (ema), 09 2021. URL <https://www.investopedia.com/terms/e/ema.asp>.
- [13] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.
- [14] CoinMarketCap. Bitcoin price today, btc to usd live, marketcap and chart | coinmarketcap, 09 2021. URL <https://coinmarketcap.com/currencies/bitcoin/>.
- [15] Wikimedia Commons. File:overfitted data.png — wikimedia commons, the free media repository, 2020. URL [https://commons.wikimedia.org/w/index.php?title=File:overfitted\\_data.png&oldid=1000000000](https://commons.wikimedia.org/w/index.php?title=File:overfitted_data.png&oldid=1000000000).

[php?title=File:Overfitted\\_Data.png&oldid=427026278](https://commons.wikimedia.org/w/index.php?title=File:Overfitted_Data.png&oldid=427026278). [Online; accessed 3-October-2021].

- [16] Wikimedia Commons. File:overfitting.svg — wikimedia commons, the free media repository, 2020. URL <https://commons.wikimedia.org/w/index.php?title=File:Overfitting.svg&oldid=452480272>. [Online; accessed 3-October-2021].
- [17] Rajashree Dash and Pradipta Kishore Dash. A hybrid stock trading framework integrating technical analysis with machine learning techniques. *The Journal of Finance and Data Science*, 2(1):42–57, 2016.
- [18] Willy David Puell. Experiments on cumulative destruction, 04 2019. URL <https://woobull.com/experiments-on-cumulative-destruction/>.
- [19] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387, 2014.
- [20] Harris Drucker, Chris JC Burges, Linda Kaufman, Alex Smola, Vladimir Vapnik, et al. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.
- [21] Jason Fernando. Moving average convergence divergence (macd), 09 2021. URL <https://www.investopedia.com/terms/m/macd.asp>.
- [22] Jason Fernando. Relative strength index (rsi), 09 2021. URL <https://www.investopedia.com/terms/r/rsi.asp>.
- [23] Github. *scikit-learn-contrib/boruta\_py*, 2021. URL [https://github.com/scikit-learn-contrib/boruta\\_py](https://github.com/scikit-learn-contrib/boruta_py).
- [24] Glassnode. Glassnode - on-chain market intelligence, 09 2021. URL <https://glassnode.com/>.
- [25] GlassnodePricing. Pricing - glassnode studio, 09 2021. URL <https://studio.glassnode.com/pricing>.
- [26] Google. Un benvenuto a colaboratory, 10 2021. URL <https://colab.research.google.com/>.

- [27] Google. Scegli il piano colab adatto alle tue esigenze, 10 2021. URL <https://colab.research.google.com/signup>.
- [28] Lorenzo Govoni. Come l'algoritmo random forest migliora le previsioni degli alberi decisionali, 10 2021. URL <https://www.lorenzogovoni.com/random-forest/>.
- [29] Gradient1. File:gradient ascent (surface).png, 09 2021. URL [https://commons.wikimedia.org/wiki/File:Gradient\\_ascent\\_\(surface\).png](https://commons.wikimedia.org/wiki/File:Gradient_ascent_(surface).png).
- [30] Gradient2. File:gradient ascent (contour).png, 09 2021. URL [https://commons.wikimedia.org/wiki/File:Gradient\\_ascent\\_\(contour\).png](https://commons.wikimedia.org/wiki/File:Gradient_ascent_(contour).png).
- [31] Alex Greaves and Benjamin Au. Using the bitcoin transaction graph to predict the price of bitcoin. *No Data*, 2015.
- [32] Gtrends. Google trends, 09 2021. URL <https://trends.google.com/trends/?geo=IT>.
- [33] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.
- [34] Adam Hayes. Stochastic oscillator, 09 2021. URL <https://www.investopedia.com/terms/s/stochasticoscillator.asp>.
- [35] Adam Hayes. Simple moving average (sma), 09 2021. URL <https://www.investopedia.com/terms/s/sma.asp>.
- [36] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, Aug 1995. doi: 10.1109/ICDAR.1995.598994.
- [37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [38] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):14413–14423, 2020.

- [39] Investing.com. Perché è così difficile battere il mercato?, 01/07/2019. URL <https://it.investing.com/analysis/perche-e-cosi-difficile-battere-il-mercato-200430675>.
- [40] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.
- [41] Suhwan Ji, Jongmin Kim, and Hyeonseung Im. A comparative study of bitcoin price prediction using deep learning. *Mathematics*, 7(10):898, 2019.
- [42] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveiro, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-69781-7.
- [43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [44] Miron B. Kursa and Witold R. Rudnicki. Feature selection with the boruta package. *Journal of Statistical Software*, 36(11):1–13, 2010. doi: 10.18637/jss.v036.i11. URL <https://www.jstatsoft.org/index.php/jss/article/view/v036i11>.
- [45] Jan Lansky. Possible state approaches to cryptocurrencies. *Journal of Systems Integration*, 8, 01 2018. doi: 10.20470/jsi.v9i1.335.
- [46] Liu, Huan, Sammut, Claude and Webb, Geoffrey I. *Feature Selection*, pages 402–406. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8\_306. URL [https://doi.org/10.1007/978-0-387-30164-8\\_306](https://doi.org/10.1007/978-0-387-30164-8_306).
- [47] Gabriel Loyer. Long short-term memory: From zero to hero with pytorch, 09 2021. URL <https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/>.
- [48] Samuele Mazzanti. Boruta explained exactly how you wished someone explained to you, 03 2020. URL <https://towardsdatascience.com/>

[boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a](#). [Online; accessed 3-October-2021].

- [49] Sean McNally, Jason Roche, and Simon Caton. Predicting the price of bitcoin using machine learning. In *2018 26th euromicro international conference on parallel, distributed and network-based processing (PDP)*, pages 339–343. IEEE, 2018.
- [50] Paolo medici. Alberi di decisione, 11 2017. URL <ce.unipr.it/~medici/geometry/node104.html>.
- [51] Cory Mitchell. Williams %r definition and uses, 09 2021. URL <https://www.investopedia.com/terms/w/williamsr.asp>.
- [52] Mohammed Mudassir, Shada Bennbaia, Devrim Unal, and Mohammad Hammoudeh. Time-series forecasting of bitcoin prices using high-dimensional features: a machine learning approach. *Neural Computing and Applications*, pages 1–15, 2020.
- [53] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL <http://www.bitcoin.org/bitcoin.pdf>.
- [54] K Pearson. Notes on regression and inheritance in the case of two parents proceedings of the royal society of london, 58, 240-242, 1895.
- [55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [56] Thearasak Phaladisailoed and Thanisa Numnonda. Machine learning models comparison for bitcoin price prediction. In *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 506–511. IEEE, 2018.
- [57] Sameer S. Pradhan, Wayne H. Ward, Kadri Hacioglu, James H. Martin, and Dan Jurafsky. Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference of the North American*

*Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 233–240, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics. URL <https://aclanthology.org/N04-1030>.

- [58] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005. doi: 10.1109/TSMCC.2004.843247.
- [59] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [60] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.
- [61] Scipy. Statistics (scipy.stats), 2021. URL <https://docs.scipy.org/doc/scipy/tutorial/stats.html>.
- [62] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.
- [63] Hervé Stoppiglia, Gérard Dreyfus, Rémi Dubois, and Yacine Oussar. Ranking a random feature for variable and feature selection. *The Journal of Machine Learning Research*, 3:1399–1414, 2003.
- [64] SVM hyperplane. File:svm max sep hyperplane with margin.png, 09 2021. URL [https://commons.wikimedia.org/wiki/File:Svm\\_max\\_sep\\_hyperplane\\_with\\_margin.png](https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png).
- [65] SVM kernel. File:kernel machine.svg, 09 2021. URL [https://upload.wikimedia.org/wikipedia/commons/f/fe/Kernel\\_Machine.svg](https://upload.wikimedia.org/wikipedia/commons/f/fe/Kernel_Machine.svg).
- [66] SVM Regression epsilon. File:svr epsilons demo.svg, 09 2021. URL [https://commons.wikimedia.org/wiki/File:Svr\\_epsilons\\_demo.svg](https://commons.wikimedia.org/wiki/File:Svr_epsilons_demo.svg).
- [67] Daiki Takeuchi, Kohei Yatabe, Yuma Koizumi, Yasuhiro Oikawa, and Noboru Harada. Real-time speech enhancement using equilibrated rnn. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal*

- Processing (ICASSP)*, pages 851–855, 2020. doi: 10.1109/ICASSP40776.2020.9054597.
- [68] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005. ISBN 0321321367.
  - [69] Greg Thomson. È ora possibile acquistare automobili tesla con bitcoin, conferma elon musk, 24/03/2021. URL [https://it.cointelegraph.com/news/elon-musk-says-tesla-now-accepts-bitcoin-from-us-customers?\\_ga=2.14055283.595419961.1633957075-1636581610.1632507276](https://it.cointelegraph.com/news/elon-musk-says-tesla-now-accepts-bitcoin-from-us-customers?_ga=2.14055283.595419961.1633957075-1636581610.1632507276).
  - [70] Wikipedia. Indice di correlazione di pearson — wikipedia, l’enciclopedia libera, 2021. URL [http://it.wikipedia.org/w/index.php?title=Indice\\_di\\_correlazione\\_di\\_Pearson&oldid=122028637](http://it.wikipedia.org/w/index.php?title=Indice_di_correlazione_di_Pearson&oldid=122028637). [Online; in data 15-ottobre-2021].
  - [71] Wikipedia. Coefficiente di correlazione per ranghi di spearman — wikipedia, l’enciclopedia libera, 2021. URL [http://it.wikipedia.org/w/index.php?title=Coefficiente\\_di\\_correlazione\\_per\\_ranghi\\_di\\_Spearman&oldid=122301008](http://it.wikipedia.org/w/index.php?title=Coefficiente_di_correlazione_per_ranghi_di_Spearman&oldid=122301008). [Online; in data 22-ottobre-2021].
  - [72] Soner Yıldırım. How to best evaluate a classification model, 03 2020. URL <https://towardsdatascience.com/how-to-best-evaluate-a-classification-model-2edb12bcc587>. [Online; accessed 3-October-2021].
  - [73] Qinghe Zheng, Xinyu Tian, Nan Jiang, and Mingqiang Yang. Layer-wise learning based stochastic gradient descent method for the optimization of deep convolutional neural network. *Journal of Intelligent & Fuzzy Systems*, 37(4):5641–5654, 2019.