Politecnico di Torino

Corso di Laurea Magistrale in
Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Analisi predittiva applicata alle metriche di performance di una base dati relazionale

Relatore
Prof. Paolo Garza

Candidato
Gregorio Dinoi

Anno Accademico 2018-2019

Indice

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Vorrei ringraziare, *in primis*, chi mi ha accompagnato durante il percorso di tesi e ne ha permesso la realizzazione, ché senza di loro non avrei scritto queste pagine. Allora rivolgo un sentito *grazie* al Professore Paolo Garza e agli Ingegneri Giuseppe Caputi e Marco Zacometti, e a tutti i colleghi di Mediamente Consulting, che mi hanno accolto con rara simpatia e disponibilità.

Tutti gli altri? Non me ne vogliate se il vostro nome non comparirà nella lista dei ringraziati (d'altro canto, vi risparmio il tempo di andare a cercarlo). Non ci sarà nessuna lista, non sarebbe corretto nei confronti di chi, inevitabilmente, verrebbe dimenticato (non siete mica in pochi!) e troverei antipatica l'idea di fare classifiche. Oltretutto, sarebbe ingrato per chi il grazie più importante non può riceverlo più.

Un appunto, però, vorrei lasciarvelo: potrei anche non ricordare i nomi di tutti voi o essere, per questo, irriconoscente, ma vi sia di consolazione sapere che l'impronta lasciata da ciascuno di voi non conosce distinzioni e, soprattutto, non cede alla memoria.

Capitolo 1

Introduzione

1.1 Background

In un contesto in continua evoluzione, come quello informatico/tecnologico, i termini *IT Automation* e *Artificial Intelligence* (AI) vengono menzionati e si fanno sentire con insistenza sempre maggiore. Sebbene i due concetti possano fare l'uno a meno dell'altro, non sono da intendersi, nell'ambito di questa trattazione, in modo mutuamente esclusivo, bensì come due facce della medesima medaglia.

Il termine *Artificial Intelligence* fu coniato dall'informatico statunitense John McCarthy nel 1955, pioniere di questa scienza, che ne diede la seguente storica definizione: "the science and engineering of making intelligent machines, especially intelligent computer programs" (McCarthy; 2007). Dunque, l'idea da cui origina il concetto di Intelligenza Artificiale affonda le proprie radici nell'ambizione di poter attribuire a una macchina, intesa come sintesi tra hardware e software, capacità che sarebbero appannaggio esclusivo dell'intelletto umano.

D'altra parte, per *IT Automation* si può intendere: "the use of software and systems to perform automated processes and tasks, replacing the manual and time-consuming work of IT professionals" (VMware; 2019). In tal caso, ciò che viene messo in evidenza è l'abilità di una macchina a svolgere in completa autonomia (i.e., senza l'intervento umano) operazioni e task più o meno complessi.

Si intuisce, dunque, come dietro il concetto di automazione possa insinuarsi quello di intelligenza artificiale: le decisioni che uno strumento software, o più in generale un sistema, prende in autonomia si potrebbero basare sull'interpretazione, da parte della macchina, di una determinata situazione.

Partendo dalle considerazioni di carattere generale fatte finora, questa trattazione si focalizza in particolar modo sull'aspetto dell'apprendimento automatico (in inglese, *machine learning*), cioè la branca dell'intelligenza artificiale che si occupa di migliorare,

attraverso l'applicazione di un insieme di metodologie, le capacità di comprensione della macchina. Più in dettaglio, il lavoro di tesi si propone di indagare l'efficacia e la validità di un'analisi di tipo predittivo applicata a un contesto ben preciso: lo stato di performance relativo all'istanza di un database relazionale. Inoltre, lo studio è stato affrontato tenendo a mente come obbiettivo anche quello di porre le basi per la realizzazione futura di uno strumento in grado, oltre a comprendere lo stato del sistema, di reagire autonomamente agli eventi.

1.2 Azienda

Il lavoro di ricerca, oggetto di questa tesi, nasce dall'incontro tra l'intenzione di indagare le possibilità dell'apprendimento automatico e la proposta dell'azienda Mediamente Consulting Srl.

Mediamente Consulting Srl è una società di consulenza operante nel settore IT con il dichiarato intento di dirigere le strategie delle imprese attraverso un percorso di sviluppo e innovazione tecnologica, servendosi dei concetti e degli strumenti propri del Business Analytics e delle conoscenze di un team di comprovata esperienza.

Uno dei punti cardine su cui si fonda l'attività dell'azienda è quindi la continua ricerca e spinta all'innovazione. A tal proposito, è degno di nota il suo recente passato all'interno dell'Incubatore di Imprese Innovative del Politecnico di Torino, venendo fra l'altro insignita nel 2016 del premio "Startup dell'anno" per meriti di crescita.

Nel corso degli anni, l'azienda ha assistito alla formazione e allo sviluppo di più aree di specializzazione: Infrastruttura tecnologica, Data Integration e Management, Corporate Performance Management, Advanced Analytics, Business Intelligence.

Il lavoro di tesi è stato svolto contestualmente all'attività di IT Management condotta dalla *unit* Infrastruttura tecnologica, il cui obbiettivo è quello di fornire servizi di progettazione, gestione e monitoraggio dell'infrastruttura tecnologica e applicativa.

Dettagli maggiori sugli aspetti che definiscono il lavoro svolto dall'area di Infrastruttura tecnologica verranno forniti nel paragrafo 1.2.1.

1.2.1 Infrastruttura tecnologica

Il lavoro di tesi è stato condotto con la collaborazione del team di Infrastruttura tecnologica. Al fine di comprendere con maggior chiarezza il contesto e le motivazioni del lavoro esposto, si rende opportuno dare una descrizione più precisa di quello che è il suo ruolo all'interno dell'azienda.

L'attività svolta dal team di Infrastruttura tecnologica consiste essenzialmente nel fornire consulenza riguardo alla gestione di sistemi informatici complessi. Nello specifico, i servizi erogati spaziano su più campi operativi: dal *deployment* dell'infrastruttura all'ana-

lisi delle performance, fino alla manutenzione e all'aggiornamento periodico di sistemi software.

Uno dei punti di forza del servizio proposto è senza alcun dubbio da individuare nella padronanza dei prodotti e delle tecnologie della famiglia Oracle, su cui l'azienda ha investito molto in termini di conoscenze, fino a guadagnarsi un'ottima reputazione presso i propri clienti.

Il lavoro di tesi è strettamente correlato all'attività di *Application Management Services* (anche nota con l'acronimo AMS) svolta dall'azienda. In linea generale, il servizio di AMS può essere descritto nei seguenti termini: "[…] a well-defined process and use of related tools to detect, diagnose, remedy and report the service quality of complex business transactions to ensure that they meet or exceed end-users' performance measurements relate to how fast transactions are completed or information is delivered to the end user […]" (Wikipedia contributors; 2018).

Appare evidente, dunque, come non sia semplice dare una definizione puntuale di tale attività, ma che dipenda in qualche modo dalle specifiche scelte operative di ciascuna azienda. Nell'erogazione del servizio di AMS, Mediamente Consulting si avvale di diversi strumenti e, soprattutto, delle competenze e del bagaglio di esperienze maturate dal team di lavoro, nonché di una struttura organizzativa che punta all'ottimizzazione delle risorse a disposizione. È proprio quest'ultimo punto che si configura di fondamentale importanza: la quantità di clienti cui fornire supporto, se rapportato alla 'dimensione' dell'azienda, richiede una certa efficienza nello svolgimento dei processi operativi.

Lo studio affrontato si propone appunto di intervenire su questo aspetto, concentrandosi, in particolare, sull'analisi delle performance di un sistema ben preciso. Nel capitolo 2 verrà data una spiegazione più esaustiva del problema affrontato, insieme alla presentazione del caso specifico preso in esame.

Capitolo 2

Problema affrontato

2.1 Presentazione del problema

Il monitoraggio di sistemi informatici, da intendersi come sintesi dell'attività di supporto e di risoluzione dei problemi, costituisce uno dei punti di maggior valore del business aziendale. L'esigenza di fornire un servizio di qualità, rafforzando così la propria posizione nel settore, richiede un impiego di risorse adeguato e, se si considera il volume dei clienti, sia attuale che in prospettiva futura, e l'eterogeneità delle problematiche da affrontare, si può intuire come ciò possa rappresentare un aspetto critico. Da qui la necessità di ricercare soluzioni per migliorare, in primo luogo, l'efficienza del servizio proposto e, di conseguenza, la sua qualità.

Per apportare un miglioramento al servizio, è opportuno comprendere, innanzitutto, cosa si intenda con *problema* e quali siano il modo di operare e gli strumenti a disposizione per affrontarlo.

Con il termine problema, riferito a un sistema informatico, si intende una serie di cause che concorrono a determinare una condizione di malfunzionamento del sistema, corrispondente a una deviazione, in negativo, dei propri parametri dai valori normali, fino a causare un degrado, più o meno grave, delle prestazioni. Nella maggior parte dei casi, il verificarsi di un problema viene segnalato dall'utente che utilizza il sistema o rilevato in modo automatico da un software di monitoraggio (per esempio, tramite l'invio di una segnalazione in seguito al superamento di una determinata soglia). Tuttavia, spesso risulta difficile individuare le cause del problema, consentendo soltanto un intervento a posteriori, cioè una volta che il problema si è verificato e/o è stato percepito dall'utente.

Il lavoro di ricerca si è focalizzato proprio su tale aspetto: indagare la possibilità di realizzare uno strumento di supporto all'attività di *monitoring* e *troubleshooting*, in grado di consentire una valutazione tempestiva delle problematiche e permettere quindi un intervento *proattivo*.

La soluzione al problema affrontato si basa sull'idea seguente: proiettare l'andamento futuro delle stato del sistema, definito attraverso la misurazione nel tempo delle sue performance, in modo da poter individuare preventivamente un eventuale degrado delle stesse prestazioni. Appare evidente come una soluzione di questo tipo richieda, innanzitutto, di comprendere quali siano i parametri significativi per la descrizione dello stato di un sistema; e, conseguentemente, richieda l'analisi dei dati raccolti e l'applicazione dei concetti di machine learning al fine di realizzare la componente predittiva.

Nello studio si è circoscritto il problema generale a un caso più semplice, riducendo la complessità delle informazioni da gestire e ottenendo in tal modo un maggior controllo sull'analisi dei dati a disposizione, quindi una maggiore capacità di valutazione dei risultati raggiunti. Nel prosieguo della trattazione verranno spiegate più in dettaglio le scelte che sono state fatte e le motivazioni che hanno portato a tali scelte, in riferimento al contesto analizzato.

In figura 2.1 viene riportato lo schema concettuale del flusso di lavoro seguito per affrontare il problema. Questo schema vuole mettere in evidenza come da una problematica di tipo generale si sia arrivati alla proposta, quindi alla progettazione, di una soluzione e come questa sia stata poi applicata a un contesto ben preciso, definito secondo i propri parametri. Per soluzione s'intende un insieme di scelte, sia di natura tecnica (e.g., prodotti e librerie software) che relative alla fase analitica (e.g., tipo di analisi, algoritmi di apprendimento automatico eccetera), per mettere in piedi un'architettura capace di realizzare l'obbiettivo preposto.

In sintesi, si è passati dall'individuazione del problema e dalla sua definizione all'interno di un contesto specifico, alla realizzazione e valutazione di una possibile soluzione.

2.2 Definizione dei parametri del problema

Nel corso di questo paragrafo, si intende fornire una definizione più formale e rigorosa del problema affrontato. È la fase preliminare di tutta l'analisi e consiste, in breve, nell'identificare i parametri del sistema che possano descriverne, dal punto di vista quantitativo, lo stato.

La definizione di stato del sistema non può ovviamente prescindere dal fatto che si stia analizzando un sistema piuttosto che un altro: ci si può riferire, per esempio, alle performance di un server o di un *web service* o, come nel caso oggetto di questa trattazione, di un database. Ciascun sistema dev'essere osservato e valutato secondo i propri parametri caratteristici, concetto che si può tradurre nell'analisi delle metriche, dei *log file* o di qualsiasi altro tipo di informazione generata dal sistema stesso o da componenti a esso associati. In altre parole, è necessario disporre di dati, siano essi valori numerici e non, che consentano di comprendere lo stato di un sistema sotto tutti i propri aspetti.

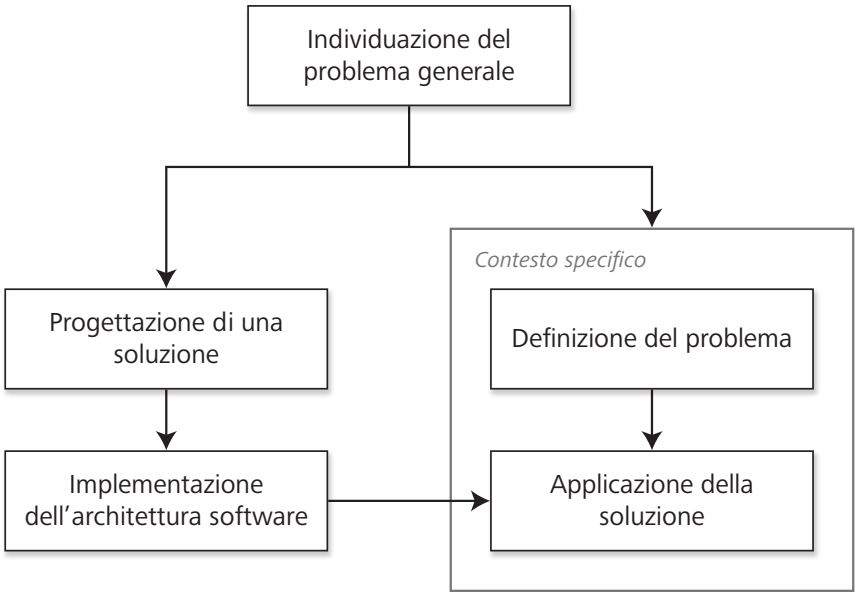


Figura 2.1: Schema concettuale del flusso di lavoro

Partendo dalle considerazioni di carattere generale fatte finora, l'interesse viene adesso rivolto alla definizione del problema all'interno di un contesto particolare. Nel paragrafo 2.1 a pagina 5 sono state indicate le motivazioni principali che hanno portato alla scelta di restringere il dominio dell'analisi a un caso ben preciso: il funzionamento dei recenti sistemi informatici implica spesso l'interazione tra varie componenti, sia hardware che software, il che comporta una maggiore complessità nell'analisi di tali sistemi. Pertanto, il caso di studio presentato riguarda esclusivamente un singolo sistema e, nello specifico, l'istanza di una base dati relazionale (*relational database management system*, RDBMS) di un cliente.

Volendo analizzare l'istanza in questione sotto l'aspetto delle prestazioni, sono stati valutati i fattori, quindi le metriche associate, che maggiormente influenzano un sistema di gestione di basi di dati: il carico di lavoro, il *throughput*, le risorse a disposizione, l'ottimizzazione del piano di esecuzione e l'accesso concorrente ai dati (Mullins; 2010).

Maggiori informazioni sul tipo di sistema analizzato vengono fornite, nei limiti della confidenzialità, nel paragrafo 2.3.

2.3 Caso di studio

La scelta di considerare un caso reale è stata dettata sostanzialmente dal bisogno di valutare l'effettiva validità della soluzione proposta. Un ambiente creato *ad hoc* avrebbe

senz'altro garantito un maggior controllo, avendo la possibilità di effettuare test mirati. Tuttavia i risultati ottenuti non sarebbero stati di alcuna garanzia in riferimento a un contesto reale.

Nello specifico, il caso oggetto di studio si riferisce alle performance relative all'istanza Oracle Database di un cliente. Per ragioni di completezza, viene riportata una breve descrizione dell'architettura su cui è ospitata l'istanza in esame, l'Oracle Database Appliance (in breve, ODA), insieme a ulteriori dettagli sulla release dell'istanza stessa e sullo strumento utilizzato per il suo *monitoring*, l'Oracle Enterprise Manager (OEM).

Oracle Database Appliance

L'Oracle Database Appliance è un sistema ingegnerizzato Oracle che si propone di offrire una soluzione semplice, ottimizzata e a costi ridotti per basi di dati e applicazioni Oracle. L'ODA viene distribuito come un sistema integrato di funzionalità software, di computazione, di rete e di storage.

In sintesi, l'ODA è composto da uno o più server Oracle Linux, connessi tra loro, e da un'unità di memorizzazione e permette l'esecuzione di database a istanza singola o in modalità cluster. Le specifiche tecniche, ovviamente, variano a seconda del modello che si sta considerando e del tipo di esigenza per cui viene utilizzato: per avere maggiori informazioni a riguardo si rimanda alla documentazione Oracle.

Oracle Database

Il target analizzato è l'istanza di un database relazionale, nella fattispecie un Oracle Database 12c Standard Edition. La versione del sistema non costituisce affatto un aspetto di secondo piano: l'utilizzo di una versione, piuttosto che un'altra, implica la possibilità di usufruire, o meno, di determinate funzionalità (Burleson; 2016). In aggiunta, Oracle permette l'integrazione anche di altre *feature*, indipendentemente dal tipo di edizione.

In base ai servizi e ai prodotti utilizzati, si può avere accesso a determinate informazioni, che possono differire anche in termini di qualità. Per esempio, l'Oracle Diagnostic Pack è un componente aggiuntivo che permette di gestire o esaminare informazioni di performance a un livello di dettaglio maggiore. Dunque, la possibilità di poter usufruire, o meno, di certe funzionalità influenza, indubbiamente, le operazioni di gestione di una base dati, che potrebbero risultare più o meno complesse.

Oracle Enterprise Manager

Una volta individuato il target da analizzare, sono state valutate le informazioni atte a delineare lo stato del sistema in esame.

L'Oracle Enterprise Manager (OEM) è una piattaforma centralizzata per la gestione dei vari prodotti Oracle che compongono un'infrastruttura. L'OEM mette a disposizione dell'utente una serie di componenti che consentono l'amministrazione del sistema sotto

diversi aspetti: è possibile pianificare attività sui database, anche remoti, gestire la sicurezza di accesso ai servizi, tenere traccia degli eventi, monitorare lo stato del sistema, solo per citare alcune delle funzionalità offerte.

Per quanto concerne il lavoro di tesi, particolare riguardo va dato al servizio di *monitoring* fornito dall'Oracle Enterprise Manager. Esso memorizza le informazioni di stato dell'ambiente gestito all'interno di un *repository*, costituito da un insieme di tabelle e viste.

I dati utilizzati nell'analisi sono stati appunto estratti da una delle numerose viste del *repository*: si fa riferimento, in particolare, alla vista *GC_METRIC_VALUES*, in cui vengono memorizzati periodicamente i valori relativi alle metriche dei diversi target monitorati.

2.3.1 Requisiti del problema

Nella fase di progettazione della soluzione, argomento trattato nel capitolo 3, bisogna tener conto di alcuni aspetti molto importanti, tra i quali la tipologia di dati da analizzare, la quantità e i tempi di raccolta ed elaborazione degli stessi. Tutti questi aspetti sono strettamente correlati con la scelta della fonte di informazioni, che, come spiegato precedentemente, è gestito dal *tool* di monitoraggio Oracle.

Per quanto riguarda la tipologia di informazioni da analizzare, va precisato che si tratta esclusivamente di valori numerici, corrispondenti alle metriche di performance del database, che nella maggior parte dei casi esprimono un numero di operazioni al secondo (e.g., letture consistenti, *commit*, scritture su disco ecc.) o misure percentuali. Questi dati vengono raccolti con una certa frequenza (in genere dell'ordine dei minuti [1]) dall'OEM e memorizzati nel proprio *repository*. Inoltre, il numero di metriche da analizzare determina la quantità di record da estrarre dalla tabella, quindi i tempi di risposta.

Le considerazioni appena fatte sono legate indubbiamente al tipo di analisi che si intende effettuare sui dati: se il modello utilizzato richiede in input l'ultima osservazione disponibile, è necessario interrogare la vista con una frequenza comparabile a quella di campionamento della metrica in esame, di modo da avere a disposizione, appunto, l'ultimo campione. Da un punto di vista più pratico, se la frequenza di campionamento per ciascuna metrica è pari a 10 minuti, allora sarà necessario ottenere l'ultimo valore entro i successivi 10 minuti. Appare evidente, dunque, come un livello di dettaglio maggiore (e.g., valori campionati ogni minuto) possa imporre dei vincoli molto stringenti sui tempi di risposta, in quanto il sistema dovrebbe reagire e fornire un output all'interno di una finestra temporale più ridotta.

[1] È possibile modificare la frequenza di rilevazione per ciascuna metrica. Tuttavia, nello studio svolto, si è tenuto conto dei valori impostati.

In aggiunta, bisogna tener conto anche della quantità di informazioni da estrarre per ogni richiesta effettuata: nel caso in esame, i record richiesti corrispondono ai valori delle metriche di interesse (circa 50) per un dato istante di tempo. Quindi, il volume di dati da gestire risulta essere piuttosto ridotto. Maggiori dettagli sul dataset analizzato e sulle problematiche relative verranno comunque forniti nel paragrafo 4.2.1 a pagina 29.

Un altro fattore da tenere in considerazione riguarda il tempo di esecuzione del modello, ovvero il tempo necessario affinché vengano elaborati i dati di input e sia generato il risultato richiesto (e.g., la predizione di una certa metrica). Dunque, il modello deve essere valutato sia dal punto di vista dell'accuratezza, cioè della bontà del risultato prodotto, che da un punto di vista più pratico, inerente ai tempi di risposta del modello stesso. Ammettendo, per ipotesi, di riuscire a definire il modello più accurato possibile, questo non sarebbe di alcuna utilità ai fini applicativi nel caso in cui richiedesse un tempo di esecuzione troppo elevato.

I vincoli finora descritti si riferiscono nello specifico alla componente attiva della soluzione, cioè alla raccolta delle informazioni da sottomettere al modello e alla sua esecuzione. La fase antecedente, che include il *preprocessing* dei dati e l'addestramento del modello, non richiede particolari condizioni, in quanto sono attività che possono essere svolte a priori. Ciò che, invece, resta da determinare è la ritenzione dell'informazione, cioè il periodo di tempo per cui i dati storici devono essere mantenuti in memoria. Questo è un aspetto difficile da stabilire in modo arbitrario ed è in qualche modo determinato dalle specifiche del modello che si intende realizzare, nonché dal tipo di problema analizzato: disporre di un numero di osservazioni notevole può influire sulle performance di un modello in modo positivo o meno.

Il capitolo 3 sarà incentrato sulla soluzione progettata per il problema identificato.

Capitolo 3

Soluzione proposta

3.1 Obbiettivi

Dall'individuazione del problema si è passati alla fase di progettazione e deployment di una soluzione in grado di approcciare il problema in esame e realizzare così un servizio di monitoraggio avanzato.

È opportuno precisare che la soluzione presentata non si pone con l'intento esclusivo di risolvere il problema in questione: le valutazioni fatte sono orientate nell'ottica di fornire una soluzione flessibile e generale, cioè in grado di essere estesa e applicata anche a tipologie di problema differenti. In altri termini, non ci si è limitati a implementare una soluzione su misura, tenendo conto cioè di un unico campo d'applicazione (come può essere, per esempio, la particolare istanza di un database), bensì si è cercato di realizzare uno strumento universale, capace di supportare anche sistemi e prodotti eterogenei, insieme ad analisi di vario tipo.

Per ovvie ragioni, la soluzione proposta verrà analizzata con particolare riguardo al problema affrontato. Tuttavia, non mancheranno riferimenti alle possibili ulteriori applicazioni che l'architettura potrebbe inglobare.

Dunque, partendo da una descrizione logica dell'architettura, si passerà alla disamina dei componenti essenziali utilizzati per realizzarla, sottolineando il contributo che ciascuno di essi apporta al funzionamento dell'architettura stessa.

3.2 Architettura generale

Nel corso di questo paragrafo si fornirà una panoramica generale dell'architettura software, descritta nei termini dei suoi componenti funzionali e delle interazioni tra di essi.

L'architettura, nel suo insieme, prevede una struttura logica ripartita su più livelli, che corrispondono, nell'ordine, ai seguenti:

- Sorgenti
- Estrazione
- Raccolta
- Analisi
- Visualizzazione

Il primo livello rappresenta, come si può intuire, le fondamenta dell'intera architettura; in altre parole, costituisce la fonte delle informazioni, le quali possono appartenere a un generico dominio (e.g., metriche di performance relative a un host, a una base dati, i log di un applicativo ecc.).

La fase di estrazione dalla sorgente restituisce il dato grezzo, il quale, affinché possa essere processato dagli strati superiori, deve formattato in modo opportuno. A tal proposito si parla di *parsing* dei dati, processo che, oltre al prelievo del dato dal flusso di informazioni, prevede anche la strutturazione del dato stesso. Una volta ottenuto il dato strutturato, questo passa attraverso un livello intermedio, quello di raccolta, che si occupa dell'immagazzinamento dei dati per futuri scopi di analisi e visualizzazione.

Se ci si pone nell'ottica di voler valorizzare le informazioni a disposizione, il livello di analisi acquisisce senza dubbio un'importanza notevole all'interno dell'intera architettura. È deputato al processing e all'elaborazione dei dati raccolti e può considerarsi il fulcro del lavoro di ricerca.

Per ultimo, lo strato di visualizzazione, che offre appunto la possibilità di rappresentare graficamente le informazioni di interesse. Si noti come la visualizzazione dei dati possa avvenire indipendentemente dal fatto che questi siano stati o meno processati.

In figura 3.1 è possibile osservare lo schema dell'architettura implementata, in cui vengono messi in evidenza il flusso dei dati attraverso i vari blocchi e i principali moduli che ne realizzano la funzione. I componenti indicati per ogni blocco fanno particolare riferimento al contesto preso in esame. Tuttavia, come più volte specificato nel corso di questa trattazione, non è da escludere la possibilità di adattare l'architettura a contesti e casi di studio differenti, che implichino un altro genere di sorgenti e/o analisi.

Nei successivi paragrafi 3.3 e 3.4, verranno presentati i prodotti e le librerie software che sono stati utilizzati per implementare la soluzione richiesta, cercando al contempo di chiarire le motivazioni che hanno portato a tali scelte.

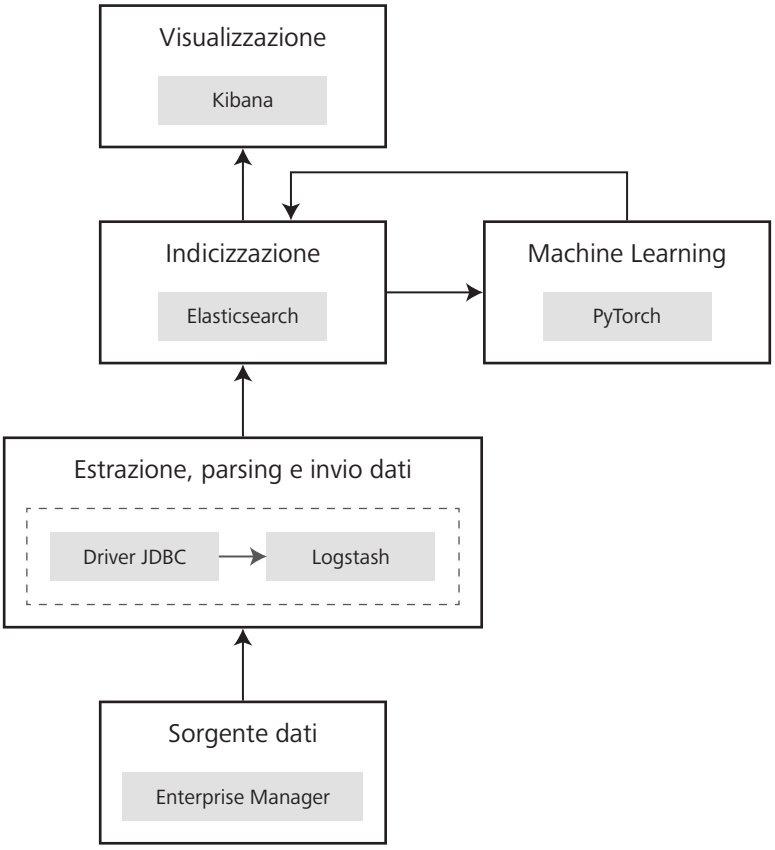


Figura 3.1: Struttura logica dell'architettura e relativi componenti

3.3 Elastic Stack

Elastic Stack è una suite di prodotti *open-source* progettata per la ricerca, l'analisi e la visualizzazione di dati in tempo reale, senza porre restrizioni sull'origine e sul formato degli stessi dati.

Elastic Stack prevede un set di componenti software, ciascuno dei quali esplica una funzione precisa e può interagire o meno con gli altri. Inizialmente, il prodotto era conosciuto come ELK Stack, dalle iniziali dei tre componenti fondamentali di cui è costituito: Elasticsearch, Logstash e Kibana. Nel corso del tempo sono stati integrati anche altri moduli (e.g., Beat), che hanno contribuito a semplificare l'uso del prodotto e ad arricchirne le possibilità. Sebbene lo *stack* sia composto da un certo numero di programmi e funzionalità, l'attenzione verrà rivolta esclusivamente ai componenti d'interesse per gli obbiettivi di questa trattazione.

In figura 3.2 vengono mostrati i software di base che appartengono a Elastic Stack,

fornendo, inoltre, un'indicazione sul livello gerarchico che ricoprono all'interno del framework. I software cui ci si riferisce sono i seguenti:

- Beat
- Logstash
- Elasticsearch
- Kibana

Per ciascun componente viene fornita una breve descrizione delle funzionalità svolte in generale e limitatamente al caso specifico.

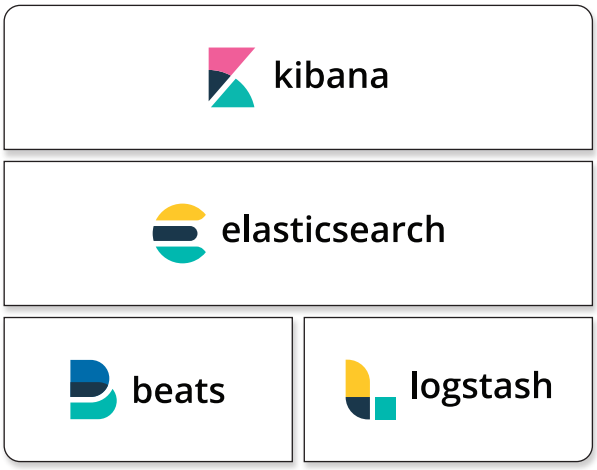


Figura 3.2: Componenti principali di Elastic Stack

3.3.1 Beat

I Beat sono degli *agent* la cui funzione principale è quella di inviare informazioni generiche, siano esse relative a un sistema o a un applicativo software, quindi senza particolari vincoli di formato, verso Logstash o Elasticsearch.

Sebbene non siano stati effettivamente utilizzati nel corso di questo lavoro, i Beat vengono comunque menzionati, in quanto rappresentano un ottimo strumento per la raccolta di dati eterogenei (e.g., i log di sistema), che, in vista di uno sviluppo futuro, potrebbero essere integrati nell'analisi predittiva.

3.3.2 Logstash

Logstash è il componente che si occupa di raccogliere dati da una o più sorgenti, applicare eventualmente dei filtri su di essi e inviarli in output verso un generico *stash* (i.e., un repository), che solitamente è rappresentato da Elasticsearch.

Logstash realizza la propria funzione attraverso la definizione di una *pipeline* composta dai seguenti stadi:

- Input
- Filtri
- Output

Il modulo di input preleva i dati da una sorgente che può essere di varia natura (e.g., file di sistema, Beat, database, socket TCP eccetera), grazie al supporto di numerosi plugin, che consentono la gestione degli eventi generati dalle sorgenti stesse.

Come spiegato nel corso del paragrafo 2.3 a pagina 7, le metriche di interesse sono memorizzate all'interno di una base dati relazionale. Pertanto, si è reso necessario l'uso del plugin di input JDBC (come evidenziato in figura 3.1) per effettuare la connessione al database e accedere ai dati. Trattandosi di una tabella relazionale, il filtro definito non prevede alcun parsing particolare, bensì effettua esclusivamente l'estrazione delle informazioni principali (i.e., nome della metrica, timestamp e valore). Queste vengono, infine, inviate, attraverso la definizione dello stage di output, a Elasticsearch.

3.3.3 Elasticsearch

Elasticsearch rappresenta, in un certo senso, il cuore di Elastic Stack, cioè lo strumento di ricerca e di analisi, nonché il collettore dei dati estratti dalle varie fonti.

In sostanza, Elasticsearch è un motore di ricerca basato sulla libreria Lucene. Attraverso interfacce web HTTP e l'uso di JSON per la formattazione dei documenti, Elasticsearch permette una semplice interazione con l'utente o altri software e la possibilità di effettuare in modo efficace ricerche full-text. Inoltre, Elasticsearch opera in un ambiente distribuito, aspetto che garantisce caratteristiche di scalabilità e resilienza, in modo del tutto trasparente per le applicazioni client con cui interagisce.

Per le finalità di questo lavoro, l'architettura costituita prevede un unico cluster di Elasticsearch composto da due nodi (i.e., due macchine distinte), ciascuno dei quali esegue la propria istanza ed è in comunicazione con l'altro.

Le informazioni inerenti ai sistemi che si intendono monitorare vengono storicizzate all'interno del cluster sotto forma di documenti JSON [1]. Nel caso di studio, queste

[1] Elasticsearch viene associato alla categoria delle basi di dati orientate ai documenti. Un documento, nella terminologia di Elasticsearch, è una struttura dati JSON, composta da coppie chiave-valore, e può essere visto come la riga di una tabella in un database relazionale.

informazioni corrispondono alle metriche di performance dell'istanza di un database e l'accesso ai relativi valori viene effettuato tramite l'esecuzione di opportune query, che restituiscono il risultato richiesto per le finalità desiderate (e.g., visualizzazione, analisi sui dati).

In sintesi, l'obbiettivo è quello di formare una base dati importante da cui attingere le varie informazioni, sfruttando l'efficienza di Elasticsearch, e su cui eseguire gli algoritmi di apprendimento automatico, per mostrare, in un secondo momento, i risultati ottenuti attraverso una piattaforma di visualizzazione, come verrà spiegato nel paragrafo 3.3.4.

3.3.4 Kibana

Kibana è la piattaforma che permette di consultare e visualizzare le informazioni di interesse sulla base dei filtri che vengono impostati.

I dati possono essere rappresentati secondo diverse tipologie di grafico, in base alle informazioni che si vogliono analizzare. È inoltre possibile la creazione di dashboard, ossia schermate in cui raccogliere le informazioni ritenute più importanti al fine di facilitarne la consultazione. Per tale motivo, questa funzionalità risulta essere di notevole utilità per chi effettua il monitoraggio di sistemi.

Nell'ottica di questo lavoro, Kibana si configura, dunque, come l'interfaccia tra sistema e utente, dove poter proiettare e visualizzare l'andamento futuro dei parametri associati al sistema monitorato, secondo la rappresentazione più appropriata (e.g., mettendo a confronto il trend storico con la proiezione futura).

3.3.5 Motivazioni

Dopo aver descritto il ruolo che ciascun componente di Elastic Stack ha rivestito all'interno dell'architettura, potrebbe risultare più evidente quale sia il fine ultimo di questo lavoro: la realizzazione di uno strumento avanzato, e al tempo stesso intuitivo, in grado di fornire supporto all'attività di gestione e manutenzione di sistemi informatici.

Elastic Stack si presenta, dunque, come un insieme di prodotti che attraverso l'interazione permettono di implementare una soluzione versatile e in certa misura universale. In altre parole, dietro la scelta di promuovere Elastic Stack c'è anche, e soprattutto, un discorso di prospettiva: la possibilità di poter analizzare e monitorare sistemi differenti, senza limitarsi a un prodotto specifico, significherebbe la realizzazione di una piattaforma unica di gestione dei sistemi IT.

Da un certo punto di vista, uno strumento del genere offrirebbe già di per sé un servizio di indubbio valore. Un accesso efficiente alle informazioni relative a un sistema, insieme a una visualizzazione immediata e intuitiva delle stesse, darebbe un contributo importante all'operatività del team di AMS.

Dalle considerazioni appena esposte, nasce l'idea di attribuire al servizio costituito valore aggiunto, attraverso l'impiego dei concetti di machine learning e lo sviluppo di

modelli predittivi da applicare ai diversi contesti. È l'aspetto più innovativo del lavoro di ricerca svolto, nonché la sfida maggiore, e si configura come parte integrante dell'intera architettura realizzata.

3.4 Linguaggio e librerie software

Lo sviluppo di uno strumento software che permetta di effettuare il processing e l'analisi dei dati non può prescindere da valutazioni, di natura più tecnica, sul tipo di linguaggio e di librerie da utilizzare: queste scelte determinano, in certa misura, le possibilità implementative di una soluzione.

La decisione di adottare un linguaggio di programmazione piuttosto che un altro è stata guidata sostanzialmente da due fattori:

- Semplicità
- Ricchezza

Nel paragrafo 3.4 vengono esposte, con particolare riguardo agli aspetti sopra citati, le motivazioni che hanno portato a scegliere Python come linguaggio di programmazione.

3.4.1 Python

Python è un linguaggio di programmazione a oggetti che coniuga molto bene semplicità d'uso (data la sua sintassi chiara e leggibile) ed espressività. È un linguaggio molto utilizzato in fase di prototipazione, proprio perché permette di realizzare un prodotto in poco tempo e con uno sforzo relativamente basso.

Occorre sottolineare un altro fattore molto determinante per la scelta del linguaggio di programmazione: la disponibilità di librerie in materia di machine learning. Python vanta di un numero considerevole di librerie e framework orientati all'analisi dei dati e all'apprendimento automatico, consolidati nel tempo grazie al contributo unificato di ricercatori e sviluppatori.

Nel corso dell'analisi, ci si è avvalsi di diverse librerie, di cui le principali vengono elencate di seguito, insieme alla versione corrispondente:

- Pandas (*0.24.2*)
- PyTorch (*0.3.0*)
- Scikit-learn (*0.20.3*)
- Statsmodels (*0.9.0*)

In questa esposizione, viene data maggiore enfasi al framework preposto per la definizione dei modelli neurali (che rappresenta una parte consistente del lavoro di ricerca), cioè PyTorch. Le altre librerie menzionate hanno contribuito da un punto di vista più analitico: sono servite, in particolare, per effettuare test e operazioni di manipolazione dei dati e per la raccolta di misure statistiche in grado di descriverne la distribuzione.

3.4.2 PyTorch

PyTorch è un framework open-source, sviluppato originariamente dal gruppo di ricerca sull'Intelligenza Artificiale di Facebook; è scritto in Python, C++ e CUDA, e viene utilizzato nell'ambito del machine learning e del deep learning. PyTorch si basa su Torch, una libreria scritta principalmente in linguaggio Lua, che fornisce un'ampia scelta di algoritmi per il deep learning.

Prima di approdare su PyTorch, sono state valutate anche altre soluzioni, in cima su tutte Tensorflow. Quest'ultimo è un framework molto diffuso, che beneficia quindi di un'ampia community. Senza fare un confronto diretto tra i due framework, è sufficiente considerare il fatto che PyTorch risulti maggiormente orientato a fini di ricerca, oltre a essere più intuitivo e facile da apprendere.

Uno degli aspetti principali di PyTorch è l'uso dei tensori, ossia vettori e matrici multi-dimensionali di numeri. Questo aspetto costituisce anche uno dei punti di forza del framework: PyTorch si integra in modo ottimale con Python (o, per usare un termine ad hoc, è molto *Pythonic*), aspetto che si riflette, per fare un esempio, nella similarità tra tensori e oggetti numpy e nella facilità con cui questi possono interagire.

È opportuno precisare che la versione di PyTorch utilizzata nel lavoro di tesi non è la più recente, bensì è stato necessario optare per una versione passata, nello specifico per la release *0.3.0*.

La scelta di risalire a una versione precedente della libreria nasce, per certi versi, dall'esigenza di potersi avvalere della potenza di calcolo delle GPU, in modo da ridurre i tempi di training e di esecuzione del modello [2]. PyTorch offre questa possibilità, sfruttando la tecnologia CUDA delle schede Nvidia. Ovviamente bisogna tener conto di un aspetto fondamentale: l'architettura hardware della scheda grafica in uso, che ne può pregiudicare, o meno, il supporto da parte della libreria.

Nelle ultime versioni di PyTorch, infatti, non è più garantito il supporto per alcuni modelli di scheda video, con un'architettura datata e capacità computazionali ridotte. Per questa ragione, si è dovuto ricorrere a una versione precedente della libreria, che supportasse il modello di scheda video a disposizione.

[2] Miglioramenti sostanziali sono stati registrati soprattutto nell'addestramento dei modelli più complessi, che comportano cioè l'elaborazione di grandi quantità di dati e l'esecuzione di operazioni onerose. Altrimenti, l'utilizzo della GPU potrebbe portare, paradossalmente, a un rallentamento del processing, dovuto all'overhead causato dal trasferimento dei dati nella memoria della GPU.

3.5 Algoritmi di machine learning

Prevedere l'evoluzione di un certo fenomeno implica fondamentalmente l'elaborazione di un modello che riesca a descriverlo e, di conseguenza, la sua applicazione ai dati rilevati nel tempo.

Nello studio svolto, sono state prese in considerazione le più recenti tecniche nel campo dell'apprendimento automatico, con l'intento di sfruttarne le potenzialità nell'analisi delle serie temporali.

Il machine learning si avvale di algoritmi che consentono a una macchina di acquisire conoscenza attraverso l'osservazione della realtà, quindi di *imparare* dall'analisi dei dati, con il fine di poter fare delle asserzioni riguardo un certo evento. Ciò è reso possibile dalla capacità di astrarre un modello generale dall'insieme dei dati di apprendimento, in grado di affrontare e risolvere correttamente nuove istanze del problema considerato.

Gli algoritmi di machine learning vengono solitamente suddivisi in tre ampie categorie, che tengono conto sia della natura del problema (e.g., classificazione, regressione, clustering) che dal tipo di informazioni disponibili (e.g., dati di input e output desiderato). Molto brevemente, le tre categorie cui si fa riferimento sono:

- Apprendimento supervisionato, che permette di determinare una regola sulla base degli input e dei corrispettivi output.

- Apprendimento non supervisionato, in cui l'algoritmo desuma una possibile struttura dai dati basandosi esclusivamente sui dati di input.

- Apprendimento per rinforzo, che prevede l'interazione con un ambiente dinamico al fine di raggiungere un obbiettivo e, più in generale, una strategia di comportamento, adattandosi ai cambiamenti mediante la distribuzione di una ricompensa o *rinforzo* (positiva o negativa).

Trattandosi, nel caso specifico, di un problema di regressione, che consiste nel prevedere il valore di una variabile numerica sulla base di uno o più predittori, o variabili indipendenti, è stato preso in considerazione un approccio di tipo supervisionato. In particolare, i modelli che sono stati definiti afferiscono a una particolare famiglia, quella delle reti neurali artificiali. Risulta necessario, quindi, chiarire cosa si intenda per *rete neurale artificiale* e quale sia il meccanismo alla base del suo funzionamento, aspetti trattati nel paragrafo 3.5.1.

3.5.1 Reti neurali artificiali

In primo luogo, viene fornita una descrizione, più o meno generica, di ciò che rappresenta una rete neurale.

> A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired by the network from its environment through a learning process.
> 2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge. (Haykin; 2009)

Partendo dalla definizione appena data, conviene soffermarsi sugli aspetti più interessanti, dal punto di vista della trattazione, che caratterizzano una rete neurale. Innanzitutto, come indica lo stesso nome, la struttura e il funzionamento di una rete neurale artificiale si rifanno alla controparte biologica, cercando quindi di emulare le capacità di comprensione che possono essere attribuite a un'intelligenza animale.

Il termine *rete* fa riferimento alla presenza delle numerose interconnessioni (le corrispondenti sinapsi) che congiungono i nodi, più propriamente detti neuroni artificiali, ovvero le unità computazionali elementari. In funzione delle sollecitazioni che riceve dall'esterno, il neurone può attivarsi o meno, emettendo, nel primo caso, un segnale in uscita, diretto all'esterno o verso altri neuroni. L'attivazione del neurone è determinata in base al fatto che l'input ricevuto sia superiore o inferiore a un certo valore di soglia, chiamata, appunto, *soglia di attivazione*. Gli impulsi cui sono sottoposti i vari neuroni della rete sono segnali modulati in relazione alla rilevanza delle informazioni che trasportano. In altre parole, i collegamenti che congiungono i neuroni sono espressi tramite pesi, che determinano, insieme alla soglia di attivazione, l'importanza del segnale trasmesso.

La conoscenza acquisita dalla rete, che si manifesta nella definizione dei pesi associati a ogni interconnessione, passa attraverso un processo di apprendimento. Facendo particolare riferimento al paradigma supervisionato, l'algoritmo che viene utilizzato per l'allenamento della rete è la retropropagazione dell'errore, in inglese *backpropagation*. Questo metodo permette di aggiornare i pesi associati a ogni connessione in modo da minimizzare una certa funzione di perdita. Per tale ragione, l'algoritmo di retropropagazione dell'errore opera in congiunzione con un metodo di ottimizzazione (per esempio, la discesa del gradiente), che permette di determinare i punti di minimo, o massimo, di una funzione.

L'algoritmo di backpropagation si compone fondamentalmente di due passi successivi:

1. *Forward pass*, in cui l'input della rete viene propagato attraverso i livelli che la costituiscono, fino a ottenere l'output corrispondente. Quindi, viene calcolato l'errore commesso dalla rete in relazione all'output desiderato.

2. *Backward pass*, che consiste essenzialmente nell'aggiornamento dei pesi della rete, propagando all'indietro l'errore commesso.

In base alla topologia, quindi al tipo di interconnessioni che sussistono tra i nodi, e all'utilizzo previsto, possono essere individuate varie classi di reti neurali. L'interesse verrà rivolto, in particolar modo, a due categorie di reti neurali artificiali, denominate rispettivamente reti neurali *feed-forward* e reti neurali *ricorrenti*.

Prima di procedere con la trattazione delle diverse tipologie di reti neurali, è opportuno discutere di un aspetto molto importante che ne determina il comportamento, ovvero il ruolo ricoperto dalla funzione di attivazione.

Funzioni di attivazione

In breve, la funzione di attivazione, in inglese *activation function*, definisce l'output di un neurone a fronte di un segnale (i.e., un insieme di valori) in ingresso. Dunque, una funzione di attivazione decide se un neurone debba essere attivato o meno. L'uso delle funzioni di attivazione serve a introdurre non linearità, permettendo quindi alla rete di imparare relazioni più complesse, che non potrebbero essere individuate da un semplice modello lineare. Inoltre, tali funzioni devono essere differenziabili, in modo da permettere l'algoritmo di retropropagazione dell'errore descritto in precedenza.

In letteratura, esistono diverse funzioni di attivazione, che si distinguono in base all'output generato e al contesto in cui vengono utilizzate. Viene riportato un elenco, che non pretende di essere esaustivo, delle funzioni di attivazione più comuni:

- Funzione lineare
- Funzione sigmoidea
- Tangente iperbolica
- Rettificatore
- Funzione softmax

Particolare interesse viene rivolto al rettificatore, quindi all'unità che ne implementa la funzione, ovvero l'*unità lineare rettificata*, meglio conosciuta con il nome ReLU (dall'inglese rectified linear unit). Questo tipo di unità è stata impiegata nell'implementazione di alcune reti proposte, per cui merita maggiore interesse.

In termini formali, il rettificatore è definito dalla seguente funzione:

$$f(x) = max(0, x) \tag{3.1}$$

dove x rappresenta il segnale di input. Dall'equazione (3.1) si deduce che il range di valori in uscita è compreso tra 0 e infinito. È possibile osservare in figura 3.3 l'andamento grafico della funzione.

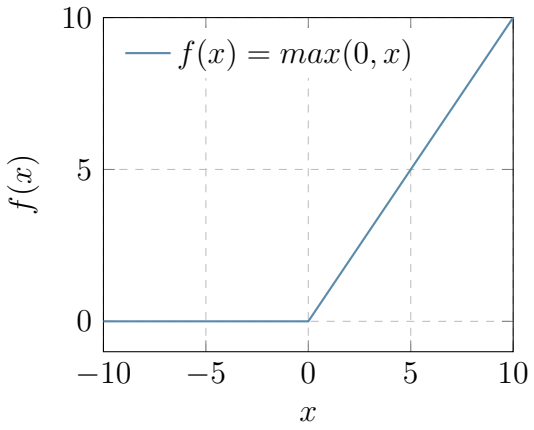


Figura 3.3: Funzione di attivazione ReLU

L'unità ReLU è molto utilizzata nelle reti neurali profonde per varie ragioni:

- La funzione prevede un valore nullo per metà del proprio dominio, facendo in modo che la metà dei neuroni della rete non siano attivi, quindi aumentandone la sparsità. Questo è un aspetto positivo, in quanto riduce l'*overfitting* in fase di addestramento.
- Rispetto ad altre funzioni di attivazione, il suo calcolo può essere realizzato in modo molto efficiente (trattandosi di un semplice confronto), comportando dunque tempi di computazione molto rapidi.
- La sua non linearità, come già spiegato in precedenza, permette di rendere la rete un approssimatore universale.

Reti neurali feed-forward

Una rete neurale di tipo feed-forward è caratterizzata da una topologia che non presenta collegamenti ciclici e in cui il flusso di informazioni viaggia in un'unica direzione, dai nodi di ingresso a quelli di uscita, passando eventualmente attraverso dei nodi intermedi, altrimenti detti *nascosti*. Facendo un parallelo con i concetti propri della teoria dei grafi, una rete neurale feed-forward può essere vista come un caso particolare di grafo aciclico diretto (DAG).

La più semplice rete neurale, appartenente alla categoria feed-forward, che può essere realizzata è il *percettrone*, che si compone esclusivamente di due layer, uno di input e uno di output. Il percettrone è privo, dunque, di layer nascosti.

L'output generato da questa semplice rete è il risultato della somma dei prodotti tra i pesi associati alle interconnessioni e i valori di input, cui viene aggiunto, qualora previsto, una fattore costante, detto *bias*.

Proposto nel 1958 da Frank Rosenblatt, il percettrone viene tradizionalmente utilizzato per l'implementazione di classificatori binari, che prevedono il *mapping* dei valori di ingresso su un output di tipo binario, in base alla funzione di attivazione prescelta. Considerando, per esempio, l'utilizzo della funzione gradino, lo spazio dei valori di ingresso verrebbe suddiviso in due classi distinte, 0 e 1. Tuttavia, la scelta di funzioni alternative, come la funzione identità, la quale non prevede un output di tipo binario, permette l'applicazione di questa rete anche a tipologie di problema che non siano, esclusivamente, di classificazione.

Il percettrone, dunque, è da intendersi come un'entità elementare, che può venire concatenata ad altre entità dello stesso tipo fino a formare una topologia più complessa. Una rete neurale artificiale composta da più percettroni è detta percettrone *multistrato*, dall'inglese *multi-layer perceptron* (MLP) e si distingue dalla sua versione più semplice, denominata *single-layer perceptron* (SLP), per la capacità di imparare relazioni non lineari.

Una rete MLP prevede, oltre agli strati di input e output, l'interconnessione di uno o più layer nascosti e, come nel caso del percettrone singolo, non ammette la presenza di cicli: i neuroni di un certo livello sono direttamente connessi con i neuroni del livello successivo.

L'architettura delle reti appena descritte è rappresentata, da un punto di vista grafico, in figura 3.5.1. Per quanto riguarda la rete SLP, figura 3.4(a), sono stati riportati in modo esplicito i pesi associati alle interconnessioni e la funzione di attivazione.

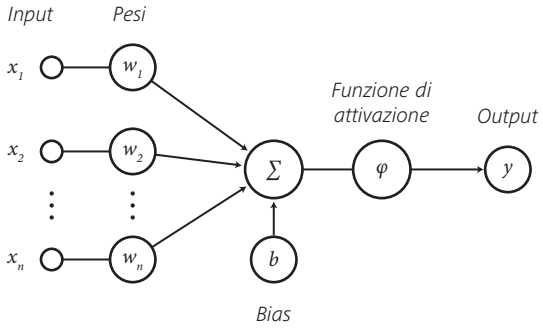
L'addestramento delle reti neurali feed-forward avviene solitamente tramite l'applicazione dell'algoritmo di retropropagazione dell'errore, argomento già discusso nell'introduzione alle reti neurali artificiali (paragrafo 3.5.1 a pagina 19).

Reti neurali ricorrenti

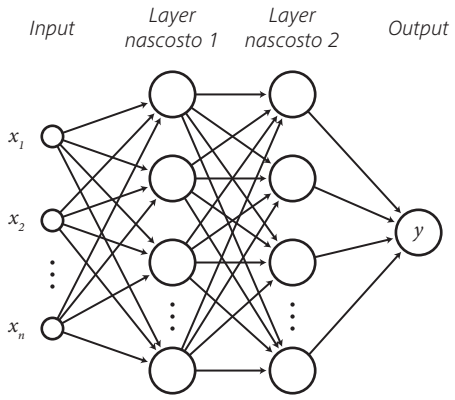
A differenza delle reti feed-forward appena trattate, le reti neurali ricorrenti sono costituite da una topologia che ammette la presenza di cicli al proprio interno, caratteristica che garantisce la persistenza delle informazioni.

Il termine *ricorrente* deriva dal fatto che la rete esegue lo stesso tipo di operazione per ogni elemento della sequenza di input, basandosi sia sull'informazione attuale che su quella passata. Difatti, una rete neurale ricorrente può essere interpretata come la concatenazione di più istanze della stessa rete, in cui ogni istanza trasmette al proprio successore l'informazione elaborata. Da ciò si può intuire come questa tipologia di rete possa ritenersi appropriata per modellare sequenze di dati (e.g., serie temporali).

L'aspetto più interessante che caratterizza una rete neurale ricorrente, cui deve molto probabilmente il successo riscosso, è la capacità di tenere traccia al proprio interno del passato, adattando in tal modo il comportamento non solo sulla base dell'informazione in ingresso, ma anche sull'esperienza acquisita. Non a caso, alle reti ricorrenti viene spesso



(a) Rete neurale *feed-forward* a singolo strato



(b) Rete neurale *feed-forward* multistrato

Figura 3.4: Topologia di due reti neurali *feed-forward*, a singolo strato e multistrato

affiancato il concetto di *memoria*, proprio perché il *modus operandi* di una rete ricorrente tende a emulare quella che è la funzione che la memoria riveste per l'uomo.

Uno dei problemi relativi all'uso delle reti neurali ricorrenti è la difficoltà nel gestire dipendenze a lungo termine, come evidenziato da Bengio et al. (1994). A tal proposito, sono stati condotti diversi studi per fronteggiare questo tipo di problematica e una possibile soluzione è stata individuata nella definizione delle cosiddette reti *long short-term memory*, o, più brevemente, reti LSTM (Olah; 2015).

Reti neurali LSTM Le reti LSTM, introdotte da Hochreiter and Schmidhuber (1997), fanno parte della famiglia delle reti neurali ricorrenti, ma presentano delle caratteristiche peculiari: sono state progettate appositamente per far fronte al problema delle dipendenze temporali a lungo termine.

I moduli che compongono una rete neurale ricorrente di tipo standard hanno una

struttura relativamente semplice, sono costituiti, per esempio, da un unico layer di rete. Una tipica unità LSTM è invece composta da quattro layer diversi, che determinano lo stato della cella e il flusso delle informazioni attraverso di essa, tramite l'azione combinata dei tre gate:

- *Input gate*
- *Forget gate*
- *Output gate*

Il funzionamento di un'unità LSTM è regolato dalle seguenti equazioni (Torch contributors; 2018):

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \tag{3.2}$$
$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \tag{3.3}$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \tag{3.4}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \tag{3.5}$$
$$c_t = f_t * c_{t-1} + i_t * g_t \tag{3.6}$$
$$h_t = o_t * \tanh(c_t) \tag{3.7}$$

dove h_t è lo stato nascosto al tempo t, c_t è lo stato della cella al tempo t, x_t è l'input all'istante t, h_{t-1} è lo stato nascosto al tempo $t-1$; i_t, f_t, o_t rappresentano rispettivamente i gate di input, forget e output, mentre g_t esprime i valori candidati per aggiornare lo stato della cella. σ è la funzione sigmoidea, la quale restituisce un valore compreso tra 0 e 1 per ogni elemento in ingresso, dove 0 corrisponde a un'informazione ininfluente, quindi da ignorare; viceversa, un valore pari a 1 individua un'informazione da tenere in 'memoria'. La funzione $\tanh$, o tangente iperbolica, produce in output valori all'interno del range $(-1, 1)$ e viene utilizzata in particolare per evitare il problema della scomparsa del gradiente, che riduce la capacità della rete di apprendere relazioni a lungo termine. Infine, il simbolo $*$ indica il prodotto di Hadamard.

Si intendono analizzare, facendo opportuni riferimenti alle formule elencate in precedenza, i passi fondamentali che realizzano il funzionamento di un'unità LSTM, al fine di esplicitare, in termini più formali, cosa rende queste reti particolarmente interessanti.

Il primo passo consiste nel determinare quali informazioni debbano essere considerate rilevanti e quali no. Questo tipo di operazione, espressa dall'equazione (3.2), tiene conto sia dell'input al generico istante di tempo t che dell'output generato in precedenza. Il risultato passa attraverso la funzione di attivazione sigmoidea, che modula il flusso di informazioni.

Lo step successivo riguarda la scelta dei valori da aggiornare e il calcolo dello stato futuro della cella. In particolare, ci si riferisce alle equazioni (3.3) e (3.4). Anche in questo caso viene utilizzata una funzione sigmoidea come filtro, mentre la tangente iperbolica

genera i valori che andranno ad aggiornare lo stato della cella. Il nuovo stato tiene conto sia dell'istante precedente che di quello corrente ed è modulato sulla base dei valori stabiliti dal forget gate e dall'input gate, come si evince dalla (3.6).

L'ultima fase decide quale dev'essere l'output della cella, cioè l'informazione da passare all'unità successiva. Tale informazione può essere vista come la versione filtrata dello stato corrente della cella. Le operazioni eseguite sono riportate in (3.5) e (3.7).

Capitolo 4

Analisi Dati e Machine Learning

4.1 Introduzione

In generale, per analisi dei dati si intende un insieme di processi, metodologie e valutazioni che contribuiscono a estrapolare dal dato grezzo informazioni utili e di valore, al fine di favorire, grazie a una maggior consapevolezza, le scelte strategiche e operative. Volendo fare un paragone, il lavoro dell'analista potrebbe essere comparato all'opera di uno scultore che servendosi delle tecniche e degli strumenti del mestiere riesce a cavar fuori da un blocco di marmo una forma del tutto nuova e avente un proprio significato.

Come più volte indicato nei precedenti capitoli, l'insieme dei dati da analizzare è costituito dalle metriche relative all'istanza di un database e l'informazione che si vuole ricavare è il comportamento futuro del sistema, dal punto di vista delle sue prestazioni. Dunque, questa fase si inserisce immediatamente dopo l'individuazione degli indici di interesse e la raccolta dei valori associati, i.e. il dataset oggetto di studio.

Bisogna tener presente che il processo di analisi non dev'essere fine a se stesso, bensì deve porre le basi per la successiva fase di studio e applicazione dei concetti di machine learning. A tal fine, l'attività di analisi prevede una serie di valutazioni da fare e di metodologie da seguire: è un percorso che dall'osservazione del fenomeno porta alla definizione e implementazione di un modello che possa descriverlo nel miglior modo possibile. Pertanto, è opportuno, se non necessario, considerare questa fase come l'unione e l'interazione di più sotto-fasi.

Si elencano, di seguito, quelli che possono essere considerati i passi fondamentali che hanno costituito il processo di analisi:

- Raccolta e strutturazione dei dati

- Ispezione dei dati
- Trasformazione dei dati
- Definizione del modello
- Addestramento del modello
- Applicazione e verifica del modello

Nel corso di questo capitolo, verranno presentati, prima in veste teorica e poi applicati al contesto specifico, tutti quei concetti e quegli strumenti che sono stati parte integrante della fase di analisi e che hanno contribuito allo sviluppo dei modelli di apprendimento automatico.

Prima di entrare nel merito di ciascun punto del processo, è doveroso fare chiarezza sul tipo di approccio che è stato adottato ai fini dell'analisi. Nel paragrafo 4.2 si parlerà, pertanto, di serie temporali, dandone in primo luogo una definizione concettuale e spiegando successivamente i motivi che hanno portato a tale scelta.

4.2 Serie temporali

Una serie temporale, o serie storica (in inglese, *time series*), è una sequenza di valori ordinata nel tempo, riferiti a una variabile casuale rilevata ad intervalli più o meno regolari. Riprendendo la definizione data dal professore James Hamilton (1994), una serie temporale può essere intesa come una delle possibili realizzazioni del processo stocastico sottostante [1].

L'analisi di serie temporali fornisce informazioni sull'andamento nel tempo di un determinato fenomeno, e.g. i prezzi di un titolo quotato in borsa, il tasso di disoccupazione, o, come nel caso oggetto di studio, le metriche di carico di un RDBMS. Ovviamente l'intervallo temporale su cui proiettare la sequenza di valori dipende dal tipo di fenomeno osservato: l'unità di misura dell'asse temporale potrebbe essere il giorno, il mese, l'anno oppure i secondi.

Le metriche analizzate, dunque, sono state trattate come variabili appartenenti a una serie temporale. L'idea alla base è quella di sfruttare concetti e tecniche proprie dell'analisi di serie temporali, solitamente applicati a settori come quello economico/finanziario e industriale, e adattarli a un contesto diverso, come nel caso considerato in esame.

In breve, questo tipo di analisi ha come obbiettivi principali quelli di individuare e descrivere la natura che sottende il fenomeno e di effettuare delle previsioni sulla sua evoluzione futura.

[1] Un processo stocastico è una famiglia di variabili casuali il cui andamento nel tempo è descritto in termini probabilistici.

Nell'analisi delle serie storiche si fa spesso riferimento a concetti, come quello di stazionarietà, e a misure statistiche (e.g., la correlazione e autocorrelazione) che aiutano in qualche modo a comprendere e descrivere un determinato fenomeno. Dal momento che, nel corso di questo studio, ci si è avvalsi di tali concetti e strumenti, risulta opportuno fornirne una definizione, per capire appunto quale sia stato il loro contributo all'interno dell'analisi.

Prima di procedere con la spiegazione di tali concetti, conviene, tuttavia, dare qualche informazione più precisa sull'insieme di dati analizzato e sulla sua formazione.

4.2.1 Dataset

La fase preliminare dell'analisi è senza dubbio da identificare nella composizione della base dati da cui estrarre le informazioni di interesse. Il processo di formazione del dataset si è svolto in due step successivi: il primo consiste nella raccolta dei dati dalla sorgente (Elasticsearch); mentre il secondo si occupa di dare una struttura all'insieme dei dati precedentemente raccolti. La fonte dati a questo livello è rappresentata da Elasticsearch (si riveda la figura 3.1 a pagina 13), che costituisce di fatto il repository di una quantità molteplice di informazioni (e.g., file di log del sistema, metriche del database). Le informazioni da analizzare vengono quindi ricercate, attraverso l'esecuzione di una query specifica, all'interno del cluster di Elasticsearch; successivamente, vengono rielaborate in modo da renderle adeguate al tipo di analisi che si intende effettuare.

Le informazioni relative a ciascuna metrica[2] sono memorizzate su Elasticsearch sotto forma di documenti: ogni documento ne contiene il valore per un dato istante di tempo. Tuttavia, volendo considerare le metriche nel loro insieme, è stata effettuata una riorganizzazione dei dati, attraverso un'operazione simile al *pivoting*, secondo una struttura tabellare, in modo da avere per ogni istante di tempo (i.e., per ogni riga) i valori relativi a ciascuna metrica.

Un altro aspetto di cui tener conto riguarda l'effettivo istante di campionamento di ogni metrica. La frequenza di rilevazione dei valori è dettata dal tool di monitoraggio dello stato del sistema (in questo caso, l'Oracle Enterprise Manager), che effettua il campionamento a una frequenza specifica per ogni tipo di metrica. La maggior parte delle metriche considerate sono rilevate con una frequenza teorica di 10 minuti. Per questo motivo, nel processo di selezione, sono state escluse le metriche rilevate con un intervallo di tempo più ampio, in modo da poter disporre del maggior numero possibile di informazioni.

Si è parlato di *frequenza teorica* per una ragione ben precisa: in condizioni reali l'intervallo può subire qualche piccola variazione, i.e., la rilevazione non avviene con preci-

[2] Per ogni metrica sono disponibili diverse informazioni, più o meno interessanti. Nel corso di questa trattazione si farà riferimento in modo particolare al valore numerico e all'istante di campionamento.

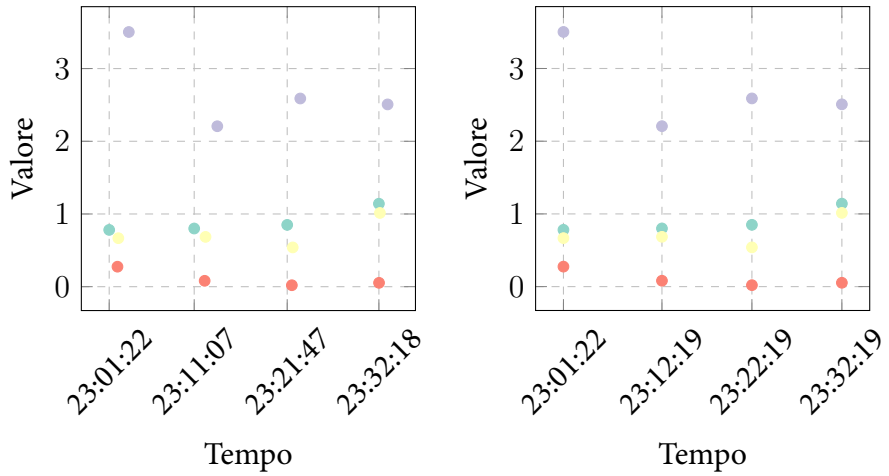
sione assoluta ogni 10 minuti. Ciò va considerato sia in rapporto alla singola metrica che per metriche differenti. È possibile osservare questo tipo di fenomeno in figura 4.1(a).

Per evitare di avere una sequenza irregolare, cioè metriche rilevate a istanti di tempo diversi, ritrovandosi valori mancanti per un dato istante e di conseguenza un dataset sparso, sono stati valutati diversi approcci. Una delle possibili soluzioni è quella di rimpiazzare i valori mancanti con una stima ottenuta attraverso un metodo statistico o analitico:

- Media, mediana
- Interpolazione
- Algoritmi di predizione

Un'altra possibilità è eliminare dall'insieme di dati gli attributi, o gli istanti di tempo, che presentano uno o più valori mancanti. Il primo caso si traduce in un'eliminazione per righe, mentre il secondo a una riduzione per colonne.

Le soluzioni appena descritte possono risultare poco opportune se si fanno le dovute considerazioni: la sostituzione dei valori mancanti introdurrebbe troppo rumore nell'analisi; d'altro canto, l'eliminazione dei record incompleti, comporterebbe una riduzione notevole, se non totale, dell'insieme dei dati.



(a) Prima dell'operazione di allineamento (b) Dopo l'operazione di allineamento

Figura 4.1: Campioni relativi a quattro metriche diverse, prima e dopo l'operazione di allineamento

Per tali ragioni, si è deciso di applicare ai dati un metodo di trasformazione alternativo: i campioni relativi a ogni metrica sono stati riallineati rispetto a un istante di tempo di

riferimento. In altre parole, si è assunto che la distanza tra le varie rilevazioni corrisponda effettivamente a 10 minuti, cioè che le metriche siano campionate a intervalli regolari. Per comprendere meglio in cosa consista questo tipo di trasformazione si osservino i grafici in figura 4.1, dove sono riportati, a titolo di esempio, i valori di alcuni campioni relativi a quattro metriche differenti prima e dopo l'operazione di allineamento.

Questo tipo di trasformazione impatta sul numero di valori mancanti presenti nel dataset. Nella tabella 4.1 vengono messi a confronto, per ciascuna metrica, il numero di record e la percentuale di valori mancanti, riferiti all'insieme di dati originale e a quello ottenuto in seguito alla trasformazione.

La base dati su cui si è lavorato raccoglie le metriche campionate in un arco temporale di 3 mesi, da Novembre 2018 a Gennaio 2019. Come riportato nella tabella 4.1, ciò equivale a disporre di circa 13000 osservazioni per metrica, se si assume come frequenza di campionamento 10 minuti [3].

Tabella 4.1: Numero di osservazioni e percentuale di valori mancanti, per ogni metrica, prima dell'operazione di allineamento (a) e dopo (b)

Metrica		Osservazioni		Valori mancanti (%)	
		a	*b*	*a*	*b*
Average Active Sessions		13 213	13 213	81,63	0,1058
Average Instance CPU (%)		13 221	13 221	81,62	0,0454
Buffer Cache Hit (%)		13 224	13 224	81,62	0,0227
User Commits (per second)		13 214	13 214	81,63	0,0983
Consistent Read Gets (per second)		13 214	13 214	81,63	0,0983
Service CPU Time (per user call) (microseconds)	SYS$USERS	13 221	13 222	81,62	0,0454
	csdb91	13 221	13 222	81,62	0,0454
	jde910	13 220	13 221	81,62	0,0529
Database CPU Time (%)		13 217	13 217	81,63	0,0756
CPU Usage (per second)		13 215	13 215	81,63	0,0907
Cursor Cache Hit (%)		11 769	11 769	83,64	11,0229
DB CPU Per Second		8 808	8 808	87,76	33,4089
DB Time Per Second		8 806	8 806	87,76	33,4241
Database Block Changes (per second)		13 214	13 214	81,63	0,0983

[3] Alcune metriche sono campionate con frequenza minore, per cui sono disponibili meno campioni.

Tabella 4.1: Numero di osservazioni e percentuale di valori mancanti, per ogni metrica, prima dell'operazione di allineamento (a) e dopo (b)

Metrica		Osservazioni		Valori mancanti (%)	
		a	*b*	*a*	*b*
Database Block Gets (per second)		13 214	13 214	81,63	0,0983
Database Time (centiseconds per second)		13 213	13 213	81,63	0,1058
Database Time Spent Waiting (%)	Administrative	13 224	13 225	81,62	0,0227
	Application	13 222	13 223	81,62	0,0378
	Commit	13 222	13 223	81,62	0,0378
	Concurrency	13 224	13 225	81,62	0,0227
	Configuration	13 223	13 224	81,62	0,0302
	Network	13 224	13 225	81,62	0,0227
	Other	13 224	13 225	81,62	0,0227
	Scheduler	13 223	13 224	81,62	0,0302
	System I/O	13 222	13 223	81,62	0,0378
	User I/O	13 223	13 224	81,62	0,0302
Service Response Time (per user call) (microseconds)	SYS$USERS	13 220	13 221	81,62	0,0529
	csdb91	13 219	13 220	81,62	0,0605
	jde910	13 220	13 221	81,62	0,0529
Enqueue Waits (per second)		13 213	13 213	81,63	0,1058
Executes (per second)		13 214	13 214	81,63	0,0983
Parse Failure Count (per second)		13 213	13 213	81,63	0,1058
Hard Parsing Time Per Second		8 806	8 806	87,76	33,4241
Hard Parses (per second)		13 214	13 214	81,63	0,0983
Host CPU Utilization (%)		13 221	13 221	81,62	0,0454
I/O Megabytes (per second)		13 213	13 213	81,63	0,1058
I/O Requests (per second)		13 214	13 214	81,63	0,0983
Library Cache Miss (%)		13 223	13 223	81,62	0,0302
Cumulative Logons (per second)		13 214	13 214	81,63	0,0983
Session Logical Reads (per second)		13 214	13 214	81,63	0,0983

Tabella 4.1: Numero di osservazioni e percentuale di valori mancanti, per ogni metrica, prima dell'operazione di allineamento (a) e dopo (b)

Metrica	Osservazioni		Valori mancanti (%)	
	a	*b*	*a*	*b*
Open Cursors (per second)	13 214	13 214	81,63	0,0983
Total Parses (per second)	13 213	13 213	81,63	0,1058
Physical Reads (per second)	13 214	13 214	81,63	0,0983
Physical Writes (per second)	13 214	13 214	81,63	0,0983
Redo Generated (per second)	13 214	13 214	81,63	0,0983
Redo Writes (per second)	13 214	13 214	81,63	0,0983
RMAN Backup/Restore Time Per Second	8 808	8 808	87,76	33,4089
User Rollbacks (per second)	13 214	13 214	81,63	0,0983
SCN Intrinsic Growth Rate (per sec)	2 196	2 196	96,95	83,3976
Soft Parse (%)	13 214	13 214	81,63	0,0983
Fast Recovery Area Space Used (%)	8 819	8 819	87,74	33,3258
User Calls (per second)	13 214	13 214	81,63	0,0983

Per reperire maggiori informazioni sulle metriche selezionate, si faccia riferimento alla documentazione Oracle (2015), in cui viene dettagliato, tra le altre indicazioni, il significato effettivo di ciascuna metrica, l'unità di misura, la frequenza di rilevazione e raccolta.

4.2.2 Osservazione dell'andamento

Il tipico approccio, nell'analisi di una serie temporale, prevede come primo passo l'osservazione dell'andamento nel tempo di un fenomeno. Nel caso specifico, il fenomeno che si vuole osservare è rappresentato dai valori relativi alle metriche di un database.

Dalla semplice osservazione dell'andamento è possibile intuire alcune informazioni utili sui valori assunti dalla metrica, come la presenza di pattern ricorrenti o di valori anomali. Questa fase permette, perciò, di valutare anche la possibilità di applicare delle

trasformazioni all'insieme dei dati di partenza, con l'obbiettivo di affinare il training del modello predittivo.

Dunque, per tracciare l'evoluzione di una metrica, ne vengono riportati i valori su un piano cartesiano, dove l'asse delle ascisse rappresenta la variabile temporale, mentre quello delle ordinate il valore della metrica nell'istante di tempo.

Si consideri, per esempio, l'andamento della metrica *Database Time* riportato in figura 4.2, riferito a un arco temporale di circa un mese. Dal grafico si distingue in modo evidente la presenza di un valore ampiamente al di fuori del range di normalità della variabile. Si può parlare in tal caso di anomalia, o *outlier*, cioè di un valore molto distante dal resto delle osservazioni. Ciò può avere un impatto significativo sull'addestramento del modello e, di conseguenza, sulle sue prestazione. Pertanto, rappresenta un aspetto, senza dubbi, molto influente, di cui occorre tener conto.

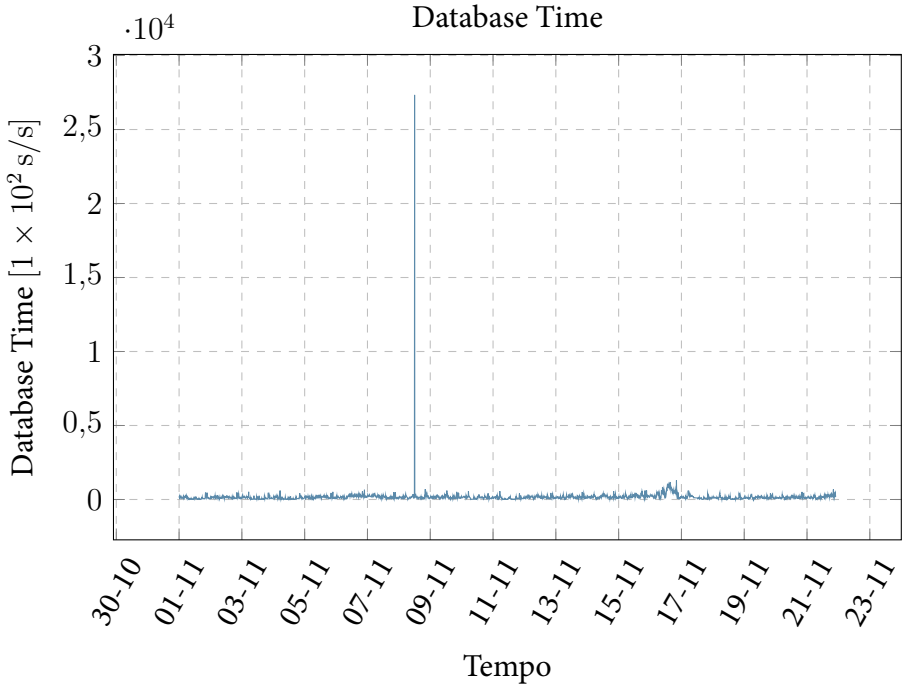


Figura 4.2: Evoluzione nel tempo della metrica *Database Time*

In figura 4.3, invece, è mostrato l'andamento relativo alla metrica *CPU Usage*. In tal caso, si vuole mettere in evidenza come i valori assunti dalla metrica nel tempo seguano un pattern più o meno definito, o, in altri termini, si ripetano con una certa periodicità. Questo aspetto caratterizza la maggior parte delle metriche considerate. Difatti, anche l'andamento della metrica tracciata in figura 4.2 presenta caratteristiche simili, sebbene non siano evidenti a causa della dinamica dei valori rappresentati: ri-

muovendo l'anomalia, il range si riduce, permettendo una visualizzazione più chiara dell'andamento.

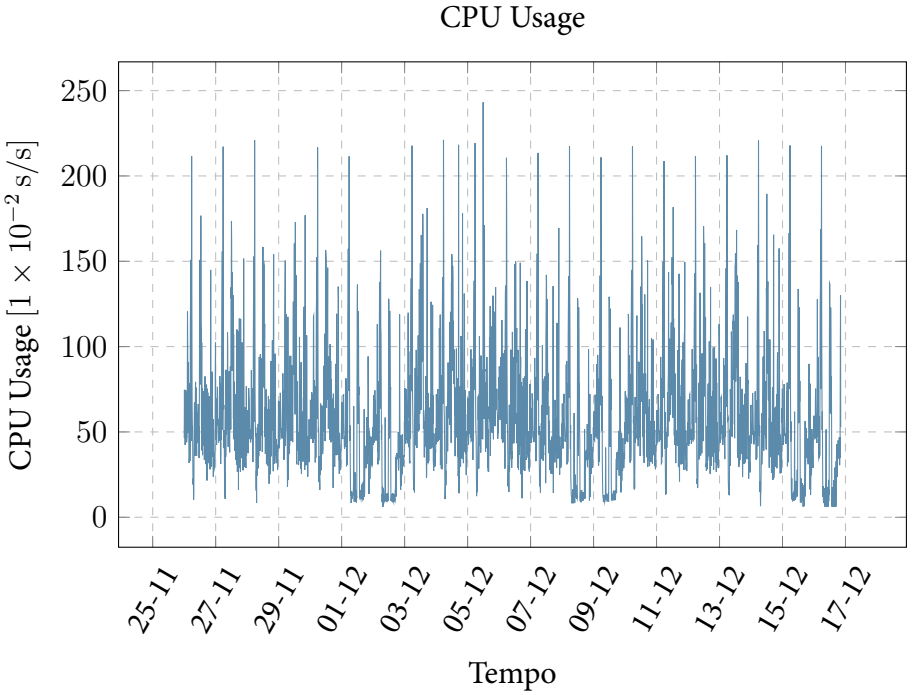


Figura 4.3: Evoluzione nel tempo della metrica *CPU Usage*

Nell'ambito delle serie temporali, un processo caratterizzato dagli stessi valori per un periodo di tempo noto e costante si dice influenzato da fattori stagionali e determina appunto una serie stagionale o periodica (in inglese, *seasonal timeseries*). Facendo riferimento alle metriche prese in esame, si può notare una certa periodicità, con cadenza giornaliera. Tale aspetto verrà ripreso più in dettaglio nel paragrafo 4.2.5, dove si parlerà di autocorrelazione.

4.2.3 Stazionarietà

Un approccio molto comune nell'analisi di una serie temporale è quello di verificare che il processo da analizzare goda o meno della proprietà di *stazionarietà*. Il motivo principale per cui si analizzano le proprietà stazionarie di un determinato fenomeno va ricercato nel fatto che un processo stazionario risulta più semplice da trattare e modellizzare. Per queste ragioni, la stazionarietà rappresenta un'assunzione comune per molte tecniche di analisi (Croarkin and Tobias; 2013).

Si possono distinguere due tipi di stazionarietà: in *senso stretto*, o forte, e in *senso largo*, o debole. Un processo stocastico si dice stazionario in senso stretto quando la sua

distribuzione di probabilità congiunta è invariante rispetto a traslazioni nel tempo. Tuttavia, questa definizione impone dei vincoli troppo restrittivi perché possa essere soddisfatta da un caso reale; per cui non viene tenuta molto in considerazione. D'altra parte, la stazionarietà debole richiede l'invarianza dei primi due momenti della distribuzione delle variabili casuali. In altre termini, un processo può considerarsi debolmente stazionario quando le proprietà di media e varianza non cambiano nel tempo e l'autocovarianza è determinata soltanto dalla distanza, o lag, tra gli indici temporali. Da un punto di vista grafico, una serie stazionaria si presenta priva di trend, componenti stagionali o, più in generale, di fluttuazioni ricorrenti nel tempo.

Nel campo delle serie temporali, molti modelli sono definiti partendo dall'assunto che il processo alla base del fenomeno analizzato abbia proprietà stazionarie. Nel lavoro di ricerca qui esposto, sono state analizzate le proprietà stazionarie per ciascuna serie temporale, al fine di poter valutare l'effetto che essa esercita sui risultati prodotti dai diversi modelli utilizzati. Dunque, risulta necessario, come primo passo, stabilire la stazionarietà di una serie ed eventualmente applicare opportune trasformazioni per renderla tale.

Per studiare la stazionarietà di una serie sono stati usati vari strumenti e metodi statistici. In primo luogo, l'osservazione dell'andamento di una variabile può fornire un'indicazione sulla stazionarietà del processo sottostante. Come analizzato nel paragrafo 4.2.2, le serie non presentano trend significativi (crescenti o decrescenti), tali da rappresentare una chiara indicazione di non stazionarietà. A riprova di quanto detto è stato eseguito il test di Dickey-Fuller aumentato, di cui si parlerà brevemente nel paragrafo 4.2.4.

4.2.4 Test di Dickey-Fuller aumentato

Il test di Dickey-Fuller aumentato viene eseguito per determinare, o fornire un'indicazione, se una serie temporale può essere o meno considerata stazionaria. Da un punto di vista più formale, il test verifica l'ipotesi nulla H_0 secondo cui la serie contiene una radice unitaria, o, in altri termini, non è stazionaria. Rifiutando l'ipotesi nulla, si sostiene che la serie non ha una radice unitaria e quindi può essere considerata stazionaria (questa viene detta ipotesi alternativa H_1).

La funzione `adfuller()`, presente nella libreria `statsmodels`, fornisce un'implementazione del test di Dickey-Fuller aumentato. Di particolare interesse, tra i valori restituiti dalla funzione, è il *p-value*, che indica la probabilità di ottenere un risultato uguale o 'più estremo' di quello osservato, supposta vera l'ipotesi nulla. Pertanto, i risultati del test sono stati valutati sulla base del *p-value*, ammettendo un livello di significatività (tipicamente denominato α) del 5% [4]. Il *p-value* viene confrontato con il valore di soglia e, nel caso in cui sia minore o uguale, si può assumere che i dati osservati non siano consistenti con l'ipotesi nulla. In sintesi:

[4] Convenzionalmente vengono utilizzati livelli di significatività del 5%, dell'1% o dello 0.1%

- Se p-value $<= \alpha$, l'ipotesi nulla può essere rifiutata e i dati non contengono una radice unitaria.

- Se p-value $> \alpha$, l'ipotesi nulla non può essere rifiutata, indicando la presenza di una radice unitaria.

Il test, eseguito sull'intero dataset, ha generato valori di *p-value* ben al di sotto del livello di significatività prescelto.

4.2.5 Autocorrelazione

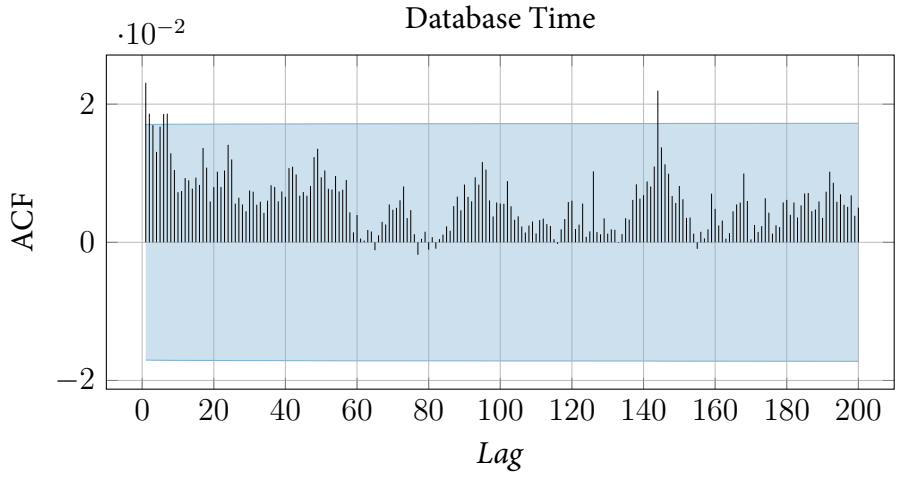
L'autocorrelazione è una misura matematica del grado di similarità tra le osservazioni di una stessa variabile o, come nel caso in esame, di una serie temporale. In altre parole, quantifica la correlazione tra una serie temporale e la sua versione traslata di un certo periodo di tempo, detto *lag*.

Come per la correlazione, anche l'autocorrelazione produce un valore nell'intervallo compreso tra -1 e 1: il primo estremo indica una forte correlazione negativa (a un incremento in una serie temporale corrisponde, in modo proporzionale, un decremento nell'altra); l'altro estremo, invece, indica una forte correlazione positiva (le serie temporali variano allo stesso modo).

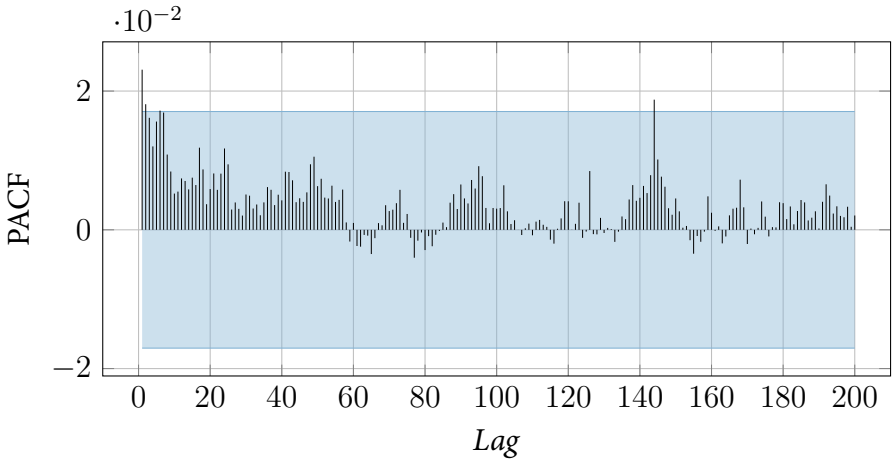
La funzione di autocorrelazione (ACF) viene generalmente utilizzata per individuare pattern all'interno dei dati. Servendosi della funzione `plot_acf()` contenuta nella libreria `statsmodels`, sono stati analizzati i grafici relativi alle funzione di autocorrelazione delle serie temporali, o metriche, analizzate.

In figura 4.4 e 4.5 vengono mostrate le funzioni di autocorrelazione riferite, rispettivamente, alle metriche *Database Time* e *CPU Usage*. L'asse delle ascisse indica il lag per cui è valutata l'autocorrelazione; l'asse delle ordinate indica, invece, il valore della correlazione. Nel grafico sono riportati i valori della funzione limitatamente a 200 lag, omettendo, per ragioni di visualizzazione, il valore al lag 0 (sempre pari a 1 per definizione). La banda di confidenza è calcolata secondo la formula di Bartlett, per un valore di α pari a 0.05.

Dall'osservazione di entrambi i grafici si può notare la presenza di valori esterni all'intervallo di confidenza, il che è indice di un'autocorrelazione significativa. In altre parole, tanto più alto è il valore della funzione a un certo istante di tempo t, tanto più forte è la correlazione tra l'osservazione al tempo t e le osservazioni agli istanti precedenti. Nello specifico, si può osservare un picco relativamente elevato in corrispondenza del lag 144, che, in relazione al dominio analizzato, equivale a 1 giorno. Ciò può essere interpretato come segue: il picco al lag 144 indica una dipendenza lineare, più o meno rilevante (in rapporto al valore assoluto della funzione in quel punto), tra ogni valore della serie e quello corrispondente a 144 punti precedenti, cioè al giorno precedente (ogni punto corrisponde a 10 minuti).



(a) Autocorrelazione

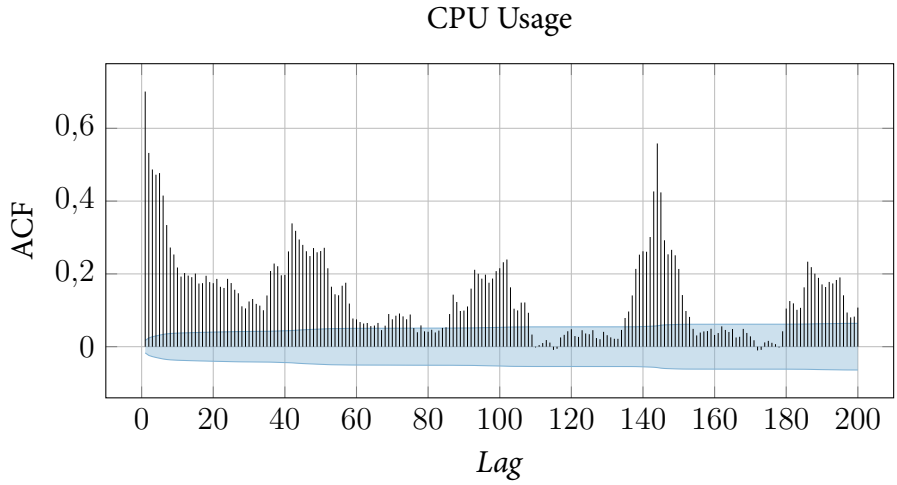


(b) Autocorrelazione parziale

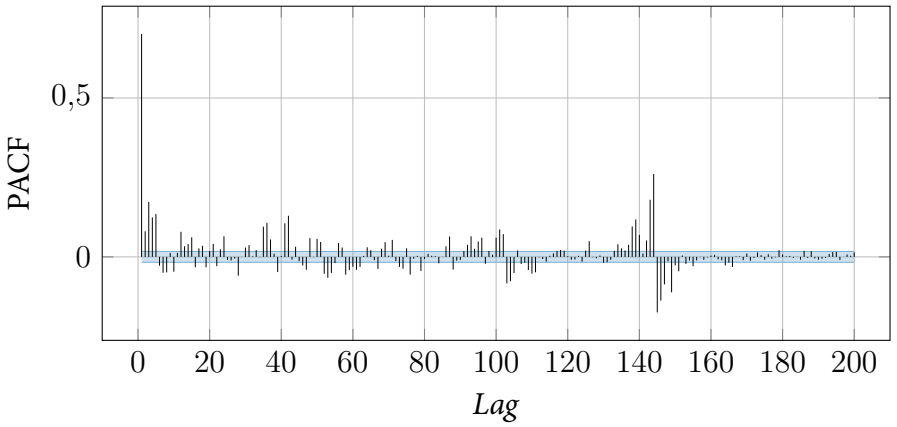
Figura 4.4: Funzioni di autocorrelazione e autocorrelazione parziale relative alla metrica *Database Time*

Tuttavia, la funzione di autocorrelazione risente del condizionamento dovuto ai valori intermedi: per esempio, la correlazione tra y_t e y_{t-k} può essere condizionata dalle variabili nel mezzo, i.e. y_{t-1}, y_{t-2} e $y_{t-(k-1)}$. Pertanto, per eliminare gli effetti delle relazioni lineari intermedie, ottenendo così una misura diretta della correlazione tra y_t e y_{t-k}, conviene riferirsi alla funzione di autocorrelazione parziale (Brownlee; 2019).

I risultati appena evidenziati possono indurre a confermare quanto anticipato nel paragrafo 4.2.2, ovvero che i valori della metrica sono caratterizzati da un andamento periodico e, più precisamente, tendono a ripetersi con frequenza giornaliera.



(a) Autocorrelazione



(b) Autocorrelazione parziale

Figura 4.5: Funzioni di autocorrelazione e autocorrelazione parziale relative alla metrica *CPU Usage*

4.2.6 Differenziazione

La differenziazione, in inglese *differencing*, è una delle possibili trasformazioni che vengono tipicamente applicate a una serie temporale non stazionaria al fine di renderla tale (Hyndman and Athanasopoulos; 2018). L'operazione di differenziazione consiste nella differenza, quindi nella misura del *cambiamento*, tra osservazioni consecutive, o distanti un certo periodo di tempo, e può essere espressa nel seguente modo:

$$y'_t = y_t - y_{t-1} \tag{4.1}$$

L'equazione (4.1) rappresenta un caso particolare di differenziazione, dove vengono considerate due osservazioni consecutive, cioè all'istante di tempo t e a quello precedente $t-1$. È possibile calcolare la differenziazione anche per *lag* superiori all'unità. Per esempio, se la serie in esame presenta una componente stagionale, si potrebbe considerare un *lag* corrispondente al periodo di stagionalità. In tal caso si parla di *seasonal differencing* e può essere calcolata secondo la formula:

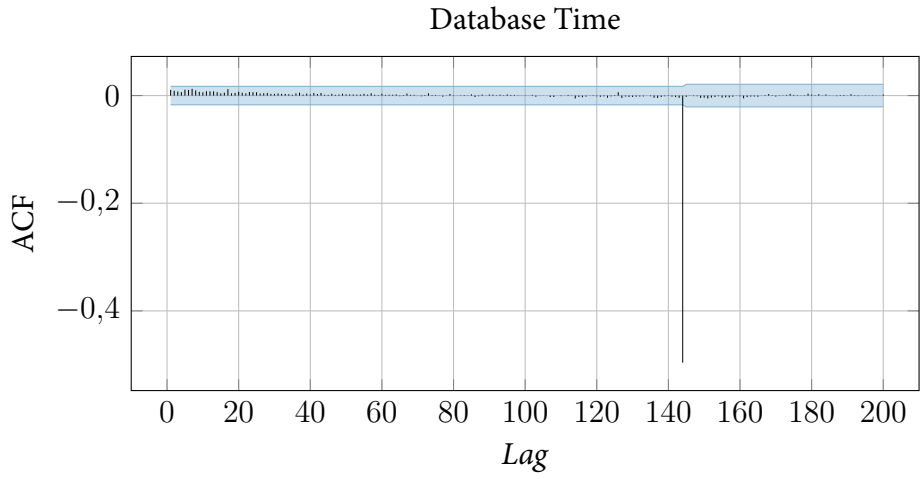
$$y'_t = y_t - y_{t-m} \tag{4.2}$$

dove m rappresenta la periodicità. In generale, dunque, il valore di *lag* per cui calcolare la differenza può essere adattato in funzione della struttura temporale propria dei dati.

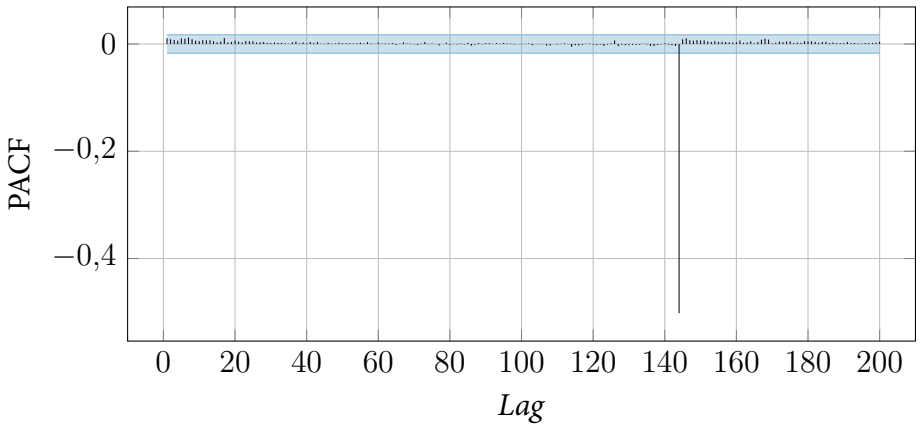
La trasformazione appena descritta è stata applicata all'insieme di metriche prese in esame, con l'obbiettivo di comprendere quali siano gli effetti di un'operazione del genere sull'analisi predittiva.

In figura 4.6 e 4.7 è possibile osservare le precedenti funzioni di autocorrelazione e autocorrelazione parziale misurate, questa volta, sulla serie differenziata. Nello specifico, la differenziazione è stata calcolata usando un *lag* pari a 144, che, secondo le considerazioni fatte in precedenza, rappresenterebbe il periodo della serie.

Come si evince dai grafici, soprattutto per determinati *lag*, sono ancora presenti autocorrelazioni significative, quindi si potrebbe desumere che anche la serie differenziata esibisce una qualche struttura temporale. Se la serie ottenuta tramite differenziazione non possiede le caratteristiche di stazionarietà desiderate, si può procedere differenziando la serie una seconda volta, aumentando quindi l'ordine di differenziazione. Tuttavia, non si è tenuto conto di questa trasformazione nell'analisi condotta, dal momento che non sembra produrre risultati significativamente migliori.

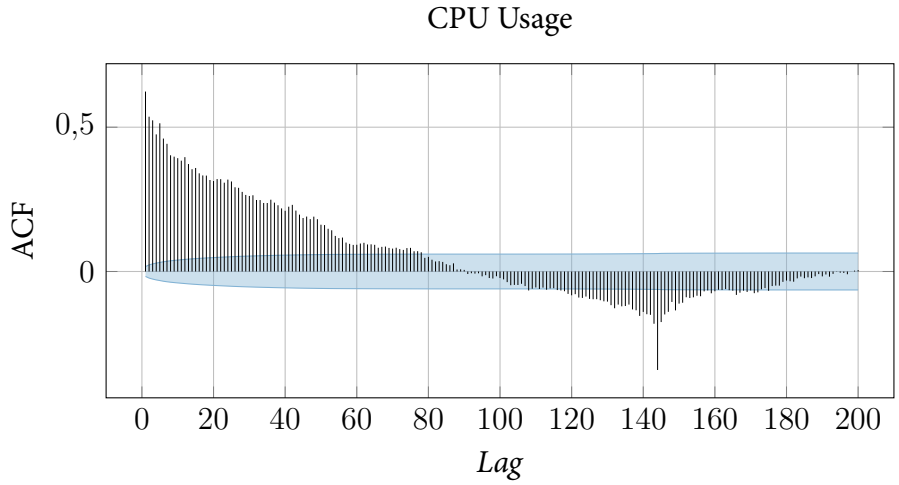


(a) Autocorrelazione

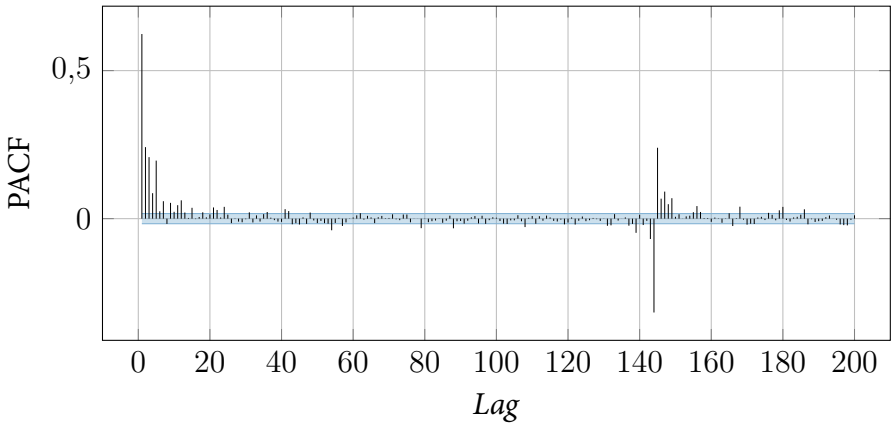


(b) Autocorrelazione parziale

Figura 4.6: Funzioni di autocorrelazione e autocorrelazione parziale relative alla serie differenziata della metrica *Database Time*



(a) Autocorrelazione



(b) Autocorrelazione parziale

Figura 4.7: Funzioni di autocorrelazione e autocorrelazione parziale relative alla serie differenziata della metrica *CPU Usage*

4.2.7 Serie temporali univariate e multivariate

Le serie temporali possono essere ulteriormente classificate in base al fatto che dipendano da una singola variabile o da variabili multiple. Nel primo caso si parla di serie *univariate* e si riferisce a una serie costituita da singole osservazioni rilevate a intervalli di tempo regolari. Per esempio, i valori relativi a una metrica, come il *Database Time* o il consumo di CPU, possono essere interpretati come una serie univariata. Un'analisi predittiva condotta su una serie di questo tipo dipenderà, dunque, esclusivamente dai valori delle osservazioni passate di quella specifica serie.

Con il termine *multivariata* ci si riferisce, invece, a una serie temporale descritta da una o più variabili. In tal caso, la serie non dipende solo dalle proprie osservazioni passate ma è influenzata, per certi versi, anche da altre variabili. L'idea di fondo è la seguente: integrare le informazioni di un determinato fenomeno con i valori di altri attributi considerati determinanti per l'evoluzione del fenomeno stesso. Si consideri, per esempio, l'utilizzo di memoria RAM: se il numero di processi in esecuzione aumenta, allora ci si può aspettare che anche il consumo di memoria cresca con esso. Pertanto, i due fattori possono considerarsi correlati.

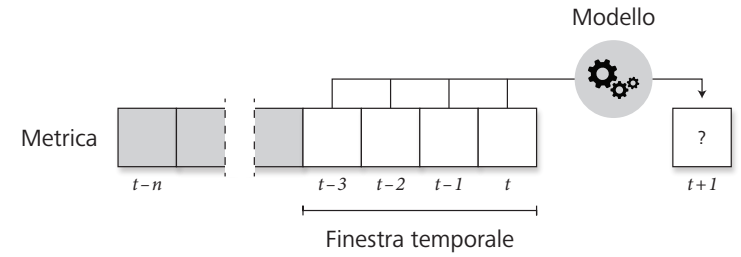
Nell'analisi svolta sono state prese in considerazione sia serie temporali univariate che multivariate, con il fine di comprendere se e quanto l'andamento di una metrica dipenda dai valori di altre metriche e come ciò possa influire sui risultati ottenuti.

I test sperimentali sono quindi stati condotti sia su serie temporali univariate, relative a una singola metrica, che multivariate, cioè composte da due o più metriche. In figura 4.8 viene rappresentato, in maniera schematica, il modo in cui il modello è applicato nei due casi. Nell'analisi delle serie multivariate si è tenuto conto della correlazione presente tra le variabili in gioco (in modo più o meno equivalente a quanto fatto con le serie univariate, con la funzione di autocorrelazione), con l'obbiettivo di determinare quale sia l'effetto di considerare variabili correlate o meno sulle prestazioni del modello.

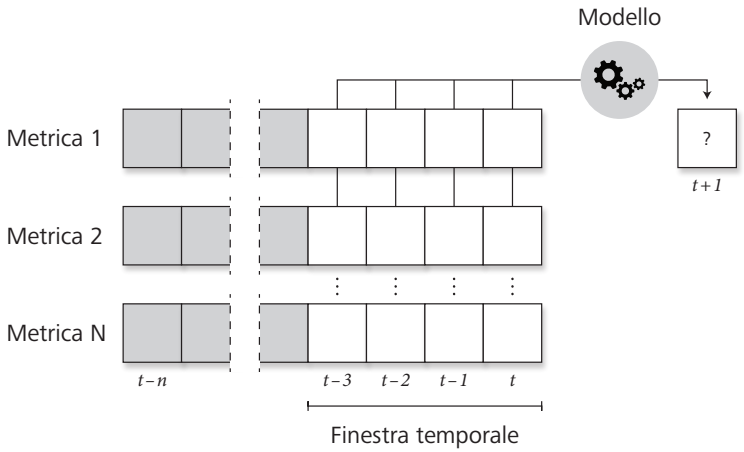
In letteratura sono stati definiti diversi indici che misurano la correlazione tra variabili. In particolare, sono stati valutati gli indici di Pearson e Spearman. Entrambi restituiscono un coefficiente compreso tra -1 e 1, che indica l'entità e la direzione della relazione. Tuttavia, i due indici hanno un significato diverso:

- L'indice di correlazione di Pearson valuta la relazione lineare che intercorre tra due variabili continue. Se sussiste una relazione lineare, allora le due variabili variano linearmente in modo proporzionale.

- L'indice di correlazione per ranghi di Spearman permette di valutare, invece, la presenza di una relazione monotona tra variabili ordinali. In tal caso, se le variabili sono correlate, allora tendono a crescere o decrescere insieme, ma non necessariamente in modo proporzionale.



(a) Serie univariata



(b) Serie multivariata

Figura 4.8: Applicazione del modello per una serie univariata e multivariata

Dal momento che gli indici appena riportati consentono di descrivere solo determinati tipi di relazioni, conviene analizzare la correlazione tra due variabili anche dal punto di vista grafico. Riportando i valori delle due variabili su un piano cartesiano si ottiene il cosiddetto grafico di dispersione, in inglese *scatter plot*. Dalla disposizione dei punti sul grafico è possibile riconoscere la presenza di una forma di correlazione tra le variabili rappresentate, che potrebbe non essere rilevata dal calcolo degli indici di Pearson o Spearman.

4.3 Modello

Dopo una prima fase di raccolta e analisi dell'insieme dei dati, lo studio si è concentrato sulla parte più propriamente predittiva, che si traduce nella definizione e implementazione di modelli in grado di predire un risultato o evento. L'elaborazione di

un modello prevede l'uso di metodi matematici e numerici per fare una stima del valore futuro di una variabile, sulla base dei dati che gli vengono forniti in ingresso. La modellazione predittiva può avvalersi, dunque, di un approccio matematico o numerico: il primo intende fornire una descrizione del fenomeno per mezzo di una o più equazioni matematiche; mentre con il secondo approccio si cerca di desumere il modello attraverso tecniche di simulazione. Ai fini di questa ricerca, sono stati presi in esame modelli basati sull'uso di reti neurali artificiali.

Prima di entrare nel merito dei dettagli implementativi e di funzionamento di una rete neurale, viene fornita una descrizione generale del processo di realizzazione del modello. In tale processo si è tenuto conto sostanzialmente dei seguenti aspetti:

- Input
- Addestramento
- Valutazione
- Applicazione

4.3.1 *Preprocessing*

Nella fase di *preprocessing* i dati vengono, appunto, processati e trasformati, o, più in generale, preparati per la fase di addestramento del modello. Le operazioni coinvolte in questa fase hanno come obbiettivo quello di migliorare il training e le prestazioni del modello.

In questa sezione vengono trattate nello specifico le operazioni relative alla normalizzazione dei dati e alla gestione delle anomalie.

Normalizzazione dei dati

La normalizzazione dei dati, in inglese *feature scaling*, è un'operazione che in genere viene effettuata durante la fase di preparazione del dataset, meglio nota come fase di *preprocessing*. Sostanzialmente, normalizzare un insieme di variabili corrisponde a limitare l'escursione dei valori relativi entro un certo intervallo predefinito.

I motivi per cui conviene normalizzare i dati sono principalmente due:

- Un range di valori molto ampio può impattare in modo negativo sul funzionamento di alcuni algoritmi, in quanto il valore di un attributo potrebbe dominare sugli altri.
- Può aiutare la convergenza, anche in termini di velocità, e la stabilità degli algoritmi di apprendimento automatico.

La libreria `scikit-learn`, per la precisione il modulo `preprocessing`, mette a disposizione diversi metodi che consentono di effettuare la riduzione in scala dei dati e che si differenziano per il modo in cui vengono stimati i parametri usati per scalare ciascun attributo. In particolare, sono state prese in considerazione le classi elencate di seguito:

- `StandardScaler`, che standardizza i dati secondo la formula $z = \frac{x-\mu}{\sigma}$, dove μ e σ rappresentano rispettivamente media e deviazione standard valutate sull'insieme di *training*.

- `RobustScaler`, che scala i dati sulla base dello scarto interquartile indicato.

Le funzioni elencate si differenziano, oltre al modo in cui trasformano i dati, anche per come reagiscono alla presenza di valori anomali, o outlier (Scikit-learn developers; 2019). In particolare, la standardizzazione effettuata tramite la classe `StandardScaler` è sensibile alla presenza di anomalie, poiché queste influiscono in modo considerevole sul calcolo di media e deviazione standard. Quindi, considerando diversi attributi, si potrebbero ottenere range di valori differenti, a causa appunto della presenza di *outlier*.

D'altra parte, la classe `RobustScaler` risulta gestire meglio la presenza di valori anomali, grazie all'uso di misure statistiche opportune, come la mediana e lo scarto interquartile. In genere, il range di valori risultante è più ampio, rispetto a quello ottenuto da altri metodi di standardizzazione; ma l'aspetto più interessante è che per attributi diversi è confrontabile. Tuttavia, questa tecnica non esclude dall'insieme dei dati le anomalie.

Gestione dei valori anomali

Molti algoritmi di machine learning possono essere influenzati dalla distribuzione e dal range di valori assunti dalle variabili di ingresso. Gli outlier possono avere un impatto significativo sia nel funzionamento degli algoritmi di machine learning, in particolare nella fase di training, inficiando l'accuratezza del modello addestrato; sia nella raccolta di statistiche sull'insieme dei dati, inducendo a trarre delle conclusioni errate.

Pertanto, nei test effettuati, è stata tenuta in conto anche la possibilità di ripulire il dataset da eventuali dati anomali. Tuttavia, se da un lato si potrebbero notare dei miglioramenti nelle performance del modello, con relativo aumento dell'accuratezza dei risultati ottenuti, dall'altro si corre il rischio di escludere dall'insieme dei dati campioni significativi per il problema. Da un altro punto di vista, la rimozione degli *outlier* potrebbe avere senso se l'obbiettivo è quello di descrivere il comportamento normale di un certo fenomeno.

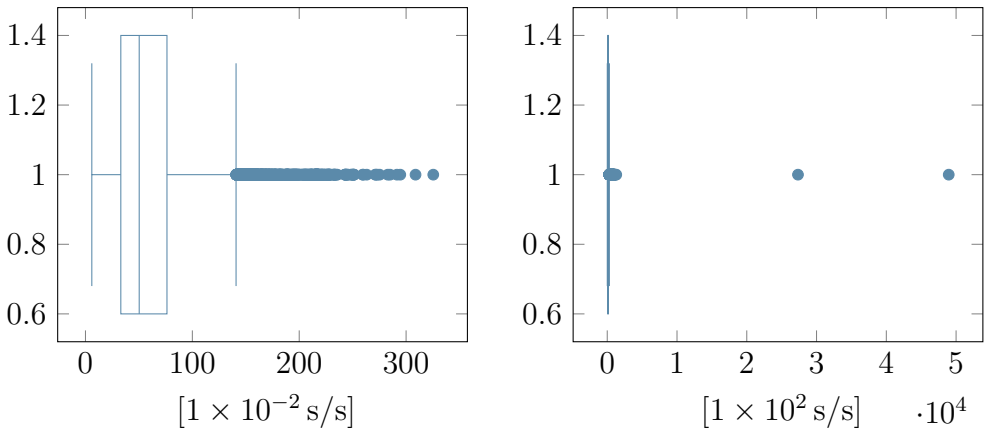
Il metodo utilizzato per identificare i valori anomali, quindi escluderli dal dataset, è il metodo dello scarto interquartile (in inglese *interquartile range* o IQR) sviluppato da John Tukey. Lo scarto interquartile può essere definito come il range tra il primo e il terzo quartile, che rappresenta, in altri termini, la porzione centrale dei valori osservati.

Quindi, lo scarto interquartile fornisce un'indicazione di quanto i valori differiscano da un valore centrale.

Un tipo di grafico che usa questo metodo per rappresentare la distribuzione dei dati è il diagramma box-plot, anche detto *box-and-whisker plot*. Difatti, il grafico presenta un rettangolo (i.e., il *box*), che rappresenta lo scarto tra primo e terzo quartile, e i cosiddetti 'baffi', cioè le estensioni che determinano se un valore debba essere considerato o meno un *outlier* (Weisstein; 2002): convenzionalmente, un valore, perché venga considerato anomalo deve essere maggiore, o minore, di 1.5 volte lo scarto interquartile.

Per chiarire in cosa consiste questo metodo, tramite anche l'ausilio della grafica, vengono mostrati in figura 4.9 i diagrammi relativi alle metriche *CPU Usage* e *Database Time*[5]. Il rettangolo rappresenta lo scarto interquartile, mentre le barre a destra e sinistra, servono a definire l'intervallo di valori 'normali'. Al di fuori di questo intervallo appaiono dei punti, che identificano i valori anomali.

Facendo riferimento ai grafici riportati, si evidenzia, soprattutto in figura 4.9(b), la presenza di valori ben al di fuori dell'intervallo centrale. Infatti, il range interquartile e le relative estensioni si riescono appena a distinguere dal grafico, a conferma del fatto che i punti anomali si trovano a una distanza considerevole.



(a) *CPU Usage* (b) *Database Time*

Figura 4.9: Box-plot relativo alle metriche *CPU Usage* e *Database Time*

In sintesi, nella gestione degli *outlier* bisognerebbe innanzitutto distinguere i casi in cui questi siano validi (i.e., propri del problema analizzato) o frutto di misurazioni errate: in quest'ultimo caso si potrebbe procedere alla loro 'correzione' senza particolari dubbi; diverso è, invece, il primo caso, in cui bisogna essere più accorti nel valutare l'esclusione o meno delle anomali, anche in relazione a quelli che sono gli obbiettivi dell'analisi.

[5] Per ragioni di visualizzazione, negli esempi riportati sono stati considerati, per ciascuna metrica, solo 5000 campioni, sebbene il dataset ne contenga un maggior numero.

4.3.2 Dati di input

Come spiegato nel paragrafo 4.2.1, il dataset in esame è costituito dai valori, ordinati nel tempo, relativi alle metriche di performance di un RDBMS. Ai fini della realizzazione del modello, e in particolare del suo addestramento, il dataset di partenza è stato suddiviso in due sottoinsiemi: il *training set* e il *testing set*. Come si può facilmente intuire dal nome, il training set contiene quei dati utili alla fase di addestramento del modello; d'altra parte, il testing set è l'insieme dei dati utilizzati in fase di valutazione del modello. In altri termini, la dimensione di ciascun insieme determina la quantità di informazioni che verranno utilizzate per stimare da una parte i parametri del modello, dall'altra l'accuratezza delle predizioni.

Tipicamente il 20% dell'intero dataset costituisce l'insieme dei dati di test; il restante 80% è destinato alla definizione del modello, quindi al training set. Nel caso di studio, la base dati di partenza è stata suddivisa secondo i suddetti valori (tramite la funzione `train_test_split` della libreria `scikit-learn`).

Ciascun set è stato ulteriormente partizionato in *chunk*, costituiti in base alla dimensione della finestra e dell'orizzonte temporale, per la successiva fase di addestramento. Ogni chunk contiene, quindi, l'insieme di osservazioni necessarie al modello per effettuare una previsione sui valori futuri.

Facendo riferimento al training del modello, per ogni step dell'algoritmo i valori che costituiscono un chunk subiscono un'operazione di *shift*, includendo, in tal modo, l'istante di tempo successivo. Iterando il processo, viene valutato l'intero insieme di training. Nel tentativo di chiarire come vengano strutturati i dati di input, viene mostrato in figura 4.10 uno schema esemplificativo, in cui è evidenziata la composizione di ciascun chunk al variare dell'istante di tempo. La fase di test avviene in modo analogo a quella appena descritta.

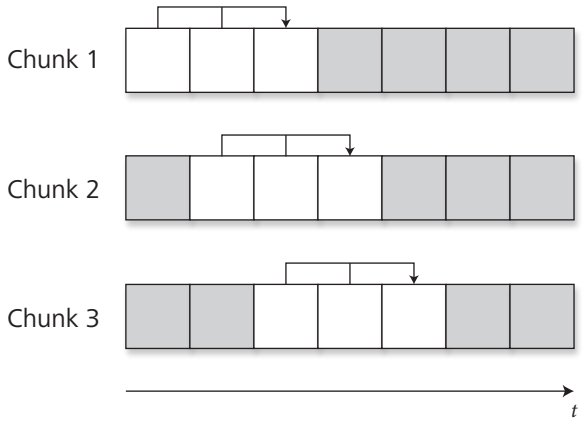


Figura 4.10: Suddivisione in chunk dell'insieme di dati di input

Nel paragrafo 4.3.3 verranno analizzati più approfonditamente gli algoritmi e i parametri che regolano l'allenamento di un modello.

4.3.3 Addestramento

La fase di addestramento prevede l'utilizzo di algoritmi la cui principale funzione è quella di calibrare, sulla base dei valori forniti in input (i.e., il training set), i parametri caratteristici del modello. Nello specifico, il *tuning* avviene tramite l'applicazione di una funzione di costo e di un metodo di ottimizzazione.

L'output generato dal processo di addestramento è un modello definito nei propri parametri caratteristici, che andrà a costituire la parte cruciale dell'intera analisi predittiva: la predizione, appunto, dei valori futuri di una variabile.

Riassumendo, i parametri coinvolti nella fase di training sono:

- Numero di epoche
- Funzione di costo
- Metodo di ottimizzazione

È bene dare una spiegazione, più o meno esaustiva, dei parametri appena elencati, in modo da chiarire la loro funzione e influenza all'interno del processo di addestramento. Inoltre, sono state effettuate diverse simulazioni variando proprio tali parametri.

Numero di epoche

Le epoche rappresentano il numero di iterazioni in cui l'algoritmo di apprendimento processa l'intero training set: un'epoca corrisponde, dunque, a una singola sessione di training. Per certi versi, è il tempo che l'algoritmo dedica all'apprendimento della natura di un fenomeno, che si traduce nell'aggiornamento dei parametri caratteristici di un modello (e.g., i pesi di una rete neurale).

Appare evidente, dunque, come questo parametro possa influire sul processo di addestramento del modello: un valore troppo basso non permetterebbe all'algoritmo di apprendere dai dati alcun pattern significativo, comportando ciò che viene definito *underfitting*; d'altra parte, un numero di epoche troppo elevato potrebbe portare al fenomeno opposto, l'*overfitting*, con la conseguenza di ottenere un modello troppo adattato ai dati di training.

Un modo per stabilire se si sia verificato il fenomeno dell'*overfitting* o *underfitting*, è quello di monitorare, per ciascuna epoca, l'evoluzione dell'errore commesso sugli insiemi di training e testing, che viene rispettivamente indicato con il termine *training loss* e *validation loss*. In particolare, nel confronto dei due andamenti si possono distinguere i seguenti casi:

- Se il *training loss* è di molto inferiore rispetto al *validation loss*, allora ciò potrebbe essere indice di *overfitting*. In altre parole, l'errore commesso sui dati di training risulta essere molto basso e questo aspetto inficia la capacità di generalizzazione della rete, rendendola quindi poco adatta a gestire nuove informazioni. Generalmente, per contrastare questo problema si riduce il numero di epoche di addestramento o si limita la dimensione della rete.

- Se, invece, il *training loss* è maggiore rispetto al *validation loss*, ci si potrebbe ritrovare in un caso di *underfitting*.

Ovviamente, la funzione di costo con cui vengono misurati questi valori dipende dalla distribuzione dei dati, per cui potrebbero presentarsi dei casi 'anomali'. Dunque, dopo aver descritto i due casi estremi, il fitting viene considerato ideale nel momento in cui *training* e *validation loss* tendono ad assumere approssimativamente lo stesso valore.

Funzione di costo

La funzione di costo è una funzione che permette di stabilire l'accuratezza delle predizioni generate dal modello. L'obbiettivo è migliorare la precisione delle ipotesi, minimizzando la distanza tra il valore atteso e quello stimato.

In letteratura, sono state definite diverse funzioni di costo che permettono di stimare i parametri di un modello, al fine di ottenere una soluzione che minimizzi l'errore. Di seguito vengono elencate le funzioni che sono state tenute in considerazione per misurare l'errore di previsione (in inglese *loss*) durante la fase di training:

- Errore medio assoluto (MAE)

- Errore quadratico medio (MSE)

- Funzione di Huber (Huber loss)

Le funzioni di costo appena riportate misurano, in base a una formula matematica, l'errore tra ogni elemento dell'input (i.e., la predizione) e il target (i.e., il valore reale). In tabella 4.2 è possibile osservare la definizione, in termini matematici, di ciascuna funzione considerata, facendo particolare riferimento alla notazione utilizzata nella documentazione di Pytorch. Osservando le espressioni corrispondenti a ogni funzione di costo, si può intuire come, in presenza dello stesso insieme di dati, possano determinare risultati differenti, andando così a influenzare in modo diverso l'addestramento del modello.

L'errore medio assoluto, in inglese *mean absolute error* (MAE), misura l'ordine di grandezza medio degli errori relativi a un insieme di predizioni, senza tener conto della direzione. È la media, calcolata su un numero di campioni, della differenza in valore assoluto tra osservazione e predizione.

MAE	$l(x,y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = \lvert x_n - y_n \rvert$ $l(x,y) = \begin{cases} \text{mean}(L) \\ \text{sum}(L) \end{cases}$ dove N è la dimensione del batch
MSE	$l(x,y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = (x_n - y_n)^2$ $l(x,y) = \begin{cases} \text{mean}(L) \\ \text{sum}(L) \end{cases}$ dove N è la dimensione del batch
Huber	$l(x,y) = \frac{1}{n} \sum_i z_i$ $z_i = \begin{cases} 0.5(x_i - y_i)^2 & \text{se } \lvert x_i - y_i \rvert < 1 \\ \lvert x_i - y_i \rvert - 0.5 & \text{altrimenti} \end{cases}$

Tabella 4.2: Funzioni di costo considerate

L'errore quadratico medio, in inglese *mean squared error*, MSE, è un altro indice di misura della discrepanza tra valori osservati e stimati. Diversamente dall'errore medio assoluto, questo indice misura l'ordine di grandezza medio dell'errore attraverso il quadrato della differenza tra il valore reale e la sua stima.

La funzione di costo di Huber è meno sensibile agli *outlier* rispetto all'errore quadratico. Come si può notare dalla sua formulazione, questa funzione è quadratica per errori piccoli (minori di 1 in valore assoluto) e lineare quando la differenza tra osservazione e predizione è maggiore.

Un altro importante parametro di cui tenere conto nella fase di addestramento è la dimensione del batch. I dati di training vengono suddivisi in sottoinsiemi, o batch, e ciascuno di essi viene processato dall'algoritmo. Se, per esempio, la dimensione del batch è pari a 128, ciò implica che verranno usati 128 campioni del set di training per stimare il gradiente dell'errore. Dunque, all'interno di un'epoca, l'algoritmo itera sull'intero set di training, ma i campioni vengono processati in gruppi.

Nel prossimo paragrafo, parlando di ottimizzazione, verrà spiegato più chiaramente come anche la dimensione del batch possa incidere sul training del modello.

Metodo di ottimizzazione

Il metodo di ottimizzazione ha come obbiettivo quello di ottimizzare una funzione nei propri parametri. Nel caso in considerazione, ciò che si vuole ottimizzare è proprio la funzione di costo.

Ai fini dell'analisi, sono stati presi in considerazione i seguenti metodi:

- Discesa stocastica del gradiente
- Adam (*Adaptive Moment Estimation*)

In generale, la tecnica di discesa secondo gradiente consiste nel determinare il minimo di una certa funzione attraverso il calcolo dell'opposto del suo gradiente, che rappresenta la direzione di massima discesa, come riportato nell'equazione (4.3).

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{4.3}$$

Nel caso in questione, si vuole minimizzare la funzione obbiettivo $J(\theta)$ rispetto ai parametri θ del modello. Il tasso di apprendimento η determina l'ampiezza dei passi compiuti per raggiungere un punto di minimo.

La discesa del gradiente è alla base dell'algoritmo di retropropagazione dell'errore usato nelle reti neurali, che consiste nell'aggiornamento dei pesi della rete al fine di ridurre l'errore.

La discesa del gradiente può essere calcolata secondo più varianti, che differiscono in base alla quantità di dati (o dimensione del batch) che contribuiscono al calcolo. In particolare, quando la dimensione del batch è pari al numero totale di campioni presenti nell'insieme di training, si parla di *batch gradient descent*. Quando il batch contiene un solo campione, l'algoritmo prende il nome di *stochastic gradient descent*. Infine, nel caso in cui la dimensione del batch è un valore compreso tra 1 e il numero totale di campioni presenti nel training set, si parla di *minibatch gradient descent*.

Sono stati condotti diversi studi che promuovono il metodo *minibatch gradient descent*, in quanto costituisce un buon compromesso in termini di performance. Si possono riassumere i motivi nei seguenti punti:

- Batch di dimensioni più o meno ridotte richiedono un utilizzo di memoria inferiore, rispetto all'intero dataset.
- L'uso di mini-batch introduce la quantità di rumore necessaria all'algoritmo per uscire da situazioni di minimi locali o punti di sella e, rispetto al metodo *stochastic gradient descent*, presenta tempi di convergenza migliori.

Adam è una tecnica di ottimizzazione che prevede il calcolo di *learning rate* adattativi per ogni parametro, a differenza del comportamento di base della discesa stocastica del gradiente, in cui il tasso di apprendimento influisce allo stesso modo per tutti i parametri del modello. In sintesi, il metodo Adam presenta i seguenti vantaggi:

- Evita che il gradiente diventi troppo piccolo, soprattutto nelle reti profonde, adattando il tasso di apprendimento.
- Non richiede un *tuning* manuale, dal momento che i tassi di apprendimento sono adattati automaticamente.

Per comprendere meglio quanto detto, vengono riportate le equazioni che regolano il funzionamento del metodo Adam:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.4}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.5}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.6}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.7}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.8}$$

$$\tag{4.9}$$

Le equazioni (4.4) e (4.5) esprimono rispettivamente le stime del primo e secondo momento dei gradienti, indicati con g_t. Essendo affetti da una tendenza verso zero, questi parametri vengono corretti dalle equazioni (4.6) e (4.7). Infine, i parametri vengono aggiornati mediante la formula (4.8).

4.3.4 Valutazione

Una volta terminato il processo di addestramento, il modello così definito è stato testato su una porzione del dataset, in modo da poter determinare le sue performance o, in altre parole, l'accuratezza dei risultati generati.

La fase di test prevede i seguenti step:

1. I dati predisposti alla fase di testing (i.e., il testing set) vengono mandati in ingresso al modello.

2. L'output generato dal modello, ossia le predizioni sui dati di test, viene confrontato con i valori reali del set di partenza tramite un indice di accuratezza.

Ottenere una stima oggettiva di quanto sia accurato il modello realizzato rappresenta un aspetto molto importante e altrettanto complesso. A conferma di ciò sono state definite diverse formule per il calcolo dell'accuratezza. Prima di entrare nel merito degli indici di accuratezza, conviene dare una definizione più formale dell'errore di predizione (in inglese *forecast error*). Esso quantifica lo scostamento tra una predizione e il valore osservato e può essere formulato nel seguente modo: $e_t = y_y - \hat{y}_t$

Per quanto concerne le metriche di accuratezza, si possono individuare diverse macrocategorie, in base al fatto che esse siano dipendenti o meno dalla scala dei dati, espresse in forma percentuale o che siano relative. Le principali misure di accuratezza che dipendono unicamente dall'errore e_t, quindi dalla scala, sono l'errore medio assoluto e la radice dell'errore quadratico medio.

Tabella 4.3: Misure di accuratezza utilizzate

Errore medio assoluto	$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}\lvert y_t - \hat{y}_t\rvert$
Errore quadratico medio	$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$
Radice dell'errore quadratico medio	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$
Errore medio assoluto percentuale	$\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\left\lvert\frac{y_t - \hat{y}_t}{y_t}\right\rvert$
Errore medio assoluto scalato	$\text{MASE} = \frac{\sum_{t=1}^{T}\lvert y_t - \hat{y}_t\rvert}{\frac{T}{T-m}\sum_{t=m+1}^{T}\lvert y_t - y_{t-m}\rvert}$

Tra le metriche che forniscono una stima percentuale dell'accuratezza di una predizione, con il vantaggio di ottenere una misura più intuitiva e indipendente dal tipo di unità, è particolarmente utilizzato l'errore medio assoluto percentuale (in inglese, mean absolute percentage error, MAPE), proprio per le ragioni appena esposte. Tuttavia, tale metrica presenta anche degli aspetti negativi: per valori di y_t prossimi a zero, l'errore percentuale tende a crescere notevolmente; inoltre, vengono penalizzati maggiormente gli errori negativi rispetto a quelli positivi (nel caso in cui $y_t < \hat{y}_t$).

Un'alternativa alle tradizionali metriche di accuratezza, che si propone di risolverne le limitazioni e i problemi connessi, è rappresentata dal *mean absolute scaled error*, o MASE. Formulato da Hyndman and Koehler (2006), questo tipo di metrica consiste sostanzialmente in "scaling the error based on the *in-sample* MAE from the naïve (random walk) forecast method". Uno degli aspetti più interessanti di questa metrica è che può essere facilmente interpretabile, dal momento che fornisce un'indicazione di quanto la predizione effettuata sia migliore rispetto a un approccio di tipo naïve.

In tabella 4.3 sono riportate le metriche di accuratezza tenute in considerazione per valutare le prestazioni dei modelli definiti.

4.3.5 Applicazione del modello

La fase che segue la definizione del modello, quindi il suo addestramento, è costituita dall'applicazione del modello stesso a un insieme di dati, al fine di ottenere i risultati

desiderati, cioè la predizione dell'evoluzione futura delle metriche in esame.

L'applicazione del modello può avvenire secondo modalità diverse, che in un certo senso dipendono anche dalla definizione specifica del modello stesso. Per esempio, il modello può essere definito e allenato in modo tale da prevedere un orizzonte temporale più o meno lungo (i.e., il numero di istanti temporali successivi a quello attuale). A tal proposito, si parla di *one-step ahead* e *multi-step ahead* per indicare i casi in cui il numero di *step* previsti sia uguale o maggiore a uno.

Il caso più interessante, considerate le specifiche del problema affrontato (e.g., la granularità delle informazioni a disposizione), è indubbiamente quello *multi-step*.

In pratica, la previsione di più istanti temporali è stata effettuata in due modi:

- Diretto
- Ricorsivo

La modalità *diretta* prevede che il modello sia in grado di valutare da sé gli istanti temporali successivi. Quella *ricorsiva*, invece, richiede che il modello venga applicato più volte sulle previsioni generate in precedenza.

Uno dei maggiori problemi che si riscontra adottando la strategia ricorsiva è il rapido degrado delle prestazioni con l'aumentare dell'orizzonte temporale, in quanto gli errori di predizione tendono ad accumularsi.

D'altro canto, la modalità *diretta* implica la definizione di un modello più complesso, proprio perché deve essere in grado di imparare le relazioni tra input e output tanto quanto quelle che intercorrono tra gli stessi valori di output. Ciò comporta una complessità anche dal punto di vista del *training*, che richiede maggior tempo, e della quantità di dati necessari.

Nel paragrafo 4.3.6 verranno presentati, da un punto di vista implementativo, i modelli che sono stati definiti.

4.3.6 Implementazione dei modelli

In fase di definizione del modello, o meglio, della rete neurale da applicare all'insieme di dati del problema (i.e., le metriche del database), al fine di proiettarne l'andamento futuro, si è deciso valutare diverse architetture, sia in termini di complessità che di funzionamento, in modo da poter indagare le capacità di ciascuna tipologia. In particolare, sono state definite tre reti neurali artificiali: due *feed-forward* e una ricorrente, nella fattispecie una rete LSTM. Per una descrizione formale del funzionamento delle suddette reti, si rimanda al paragrafo 3.5 a pagina 19, dove ne vengono trattati gli aspetti generali. In questo paragrafo, invece, le reti neurali vengono analizzate da un punto di vista più implementativo. In prima analisi, è stata realizzata una rete *feed-forward* molto elementare, costituita esclusivamente dai layer di input e output: i valori in ingresso vengono mappati, attraverso l'applicazione di una funzione lineare, su uno o più valori di uscita,

tramite la formula descritta nell'equazione (4.10), dove y è il vettore, o meglio, il tensore di uscita, x il tensore in ingresso, A è la matrice dei pesi della rete e b è l'eventuale valore di bias. Il numero dei nodi di ciascun layer, che determina anche la dimensione dei tensori, non è definito a priori, bensì può essere considerato un iperparametro.

$$y = xA^T + b \tag{4.10}$$

La seconda rete *feed-forward* definita si differenzia dalla precedente per la sua maggiore complessità, essendo costituita, in aggiunta agli strati di input e output, anche dall'interconnessione di due layer nascosti. L'output di ciascuno strato nascosto passa attraverso una funzione di attivazione, il rettificatore o *ReLU* (dall'inglese rectified linear unit). Come nel caso precedente, il numero di neuroni per ciascuno strato varia in base alla configurazione che si intende testare. Per ultima, è stata valutata una rete neurale, appartenente alla categoria delle reti ricorrenti, composta da uno o più layer LSTM e da uno strato di tipo lineare terminale, incaricato, quindi, di generare l'uscita. L'output della rete LSTM, prima di essere elaborato dallo strato lineare, viene filtrato da una funzione ReLU. Valgono le stesse considerazione fatte per le altre reti neurali: in questo caso, oltre alla dimensione della sequenza di input e output, bisogna definire il numero di layer LSTM e la dimensione degli strati nascosti.

Dal momento che l'obbiettivo è quello di tracciare l'evoluzione dello stato del sistema, i dati che le reti si aspettano in ingresso hanno un formato determinato: le rete elabora sequenze di valori che corrispondono alle osservazioni di una o più serie. La lunghezza della sequenza definisce la finestra temporale e stabilisce il numero di nodi di input della rete.

Dal punto di vista analitico, la rete descritta inizialmente cerca di stabilire una relazione lineare tra la variabile in uscita e i valori di una o più variabili di ingresso; negli altri due casi, la relazione è di tipo non lineare.

Capitolo 5

Risultati sperimentali

5.1 Introduzione ai test

Gli argomenti trattati finora, soprattutto da un punto di vista teorico, possono essere considerati, per certi versi, propedeutici alla fase di sperimentazione vera e propria.

Per l'appunto, nell'esecuzione dei vari test ci si è avvalsi degli strumenti e dei concetti esposti nel corso del capitolo 4 e sono stati valutati i risultati ottenuti secondo le diverse configurazioni dei test. Per essere più precisi, gli esperimenti svolti differiscono, principalmente, per le scelte inerenti agli aspetti elencati di seguito:

- *Preprocessing* dei dati
- Condizioni iniziali del problema
- *Iperparametri* della rete neurale
- Funzioni di costo e metodi di ottimizzazione

Come si può intuire, il numero di possibili configurazioni che possono essere valutate, al variare di ciascun parametro, è consistente e, se si considera l'insieme di metriche da analizzare, aumenta ulteriormente. Pertanto, sono stati condotti dei test mirati per un preciso sottoinsieme di metriche. In altre parole, le metriche di un database possono essere raggruppate in categorie, in relazione agli aspetti del database di cui forniscono una misura: utilizzo del disco, della CPU, consumo di memoria eccetera. Per quanto riguarda l'analisi svolta, si è deciso di esaminare con particolare riguardo l'insieme di metriche relative all'utilizzo della CPU.

Di seguito vengono specificate le metriche su cui si sono focalizzati i test:

- Service CPU Time (per user call)

- Database CPU Time (%)
- CPU Usage (per second)
- Database Time (centiseconds per second)
- Host CPU Utilization (%)

Nei prossimi paragrafi si analizzeranno le performance di ciascuna rete neurale realizzata, che, come spiegato nel paragrafo 4.3.6 a pagina 56, sono costituite da architetture differenti. Per questioni di chiarezza espositiva, si farà riferimento alle diverse reti nei seguenti termini:

- Percettrone a singolo strato, o SLP (Single-Layer Perceptron), per indicare la più semplice delle reti realizzate, cioè quella composta unicamente dagli strati di input e output.
- Percettrone multistrato, o MLP (Multi-Layer Perceptron), ovvero l'architettura che prevede, nel caso specifico, l'interconnessione di due layer nascosti.
- Rete neurale LSTM (Long Short-Term Memory), cioè la topologia che presenta al proprio interno uno o più layer di tipo LSTM.

In primo luogo, ciascun modello verrà analizzato in modo indipendente, confrontando i risultati ottenuti al variare dei parametri sopra citati. Successivamente si metteranno a paragone le prestazioni dei vari modelli, al fine di stabilire, a parità di metrica analizzata, quale topologia di rete neurale fornisce un risultato migliore, ovvero una predizione più accurata.

Inoltre, si terrà conto, oltre alla qualità del risultato generato, espresso in termini di accuratezza, secondo le metriche descritte nel paragrafo 4.3.4 a pagina 54, anche di altri fattori, quali i tempi di addestramento della rete e di esecuzione.

Prima di passare all'analisi dei risultati ottenuti per ciascun modello, conviene dare un'indicazione più precisa su come sono stati eseguiti i test: quali parametri sono stati variati, in che modo e con quale entità. Questi rappresentano, dunque, gli aspetti trattati nel paragrafo 5.2.

5.2 Configurazione dei test

Non conoscendo a priori la topologia di rete 'migliore', cioè tale da estrarre dall'osservazione dei dati un modello in grado di descrivere al meglio il processo analizzato, ma basandosi esclusivamente sui risultati e sulle intuizioni dell'analisi effettuata in precedenza, i diversi esperimenti sono stati condotti secondo un approccio che potrebbe quasi definirsi *euristico* o *trial and error*.

Come anticipato nell'introduzione a questo capitolo, i test sono stati eseguiti verificando configurazioni diverse, ossia variando in certa misura i parametri considerati significativi e valutando, caso per caso, i risultati prodotti.

Per fare un esempio, si consideri la finestra temporale, cioè il numero di osservazioni storiche di una metrica. Si è passati, dunque, dalla valutazione del caso più semplice (i.e., finestra temporale composta da un solo campione), fino a casi più complessi, sottoponendo quindi al modello sequenze di campioni via via più lunghe.

Praticamente, è impensabile effettuare degli esperimenti completamente esaustivi, che contemplino, per ogni parametro, l'intero spazio dei possibili valori, e, tanto meno, variare ciascun parametro in modo puntuale (e.g., test eseguiti su una finestra temporale che aumenta un campione alla volta): ciò richiederebbe una quantità eccessiva di tempo e buona parte dei test non porterebbero comunque a nessun risultato utile. Per queste ragione, nella variazione dei singoli parametri sono state fatte delle considerazioni più o meno ragionevoli, legate all'analisi dei dati, al problema specifico in esame e a osservazioni pubblicate da ricercatori e studiosi.

Fatte queste premesse, nel paragrafo 5.3 vengono mostrati, per ogni modello definito, i risultati dei test relativamente a una configurazione specifica. Dunque, insieme alla misura dell'accuratezza, e, ovviamente, all'indicazione della rete utilizzata, vengono trascritti anche i parametri del modello che hanno generato tale risultato, utile per comprendere come ne vengano influenzate le prestazioni in relazione alla modifica delle condizioni.

5.3 Esposizione dei risultati

I risultati verranno presentati in modo più o meno sistematico, specificando il parametro su cui viene posto il focus e mettendo in evidenza i casi particolari.

In ciascuna tabella verranno indicati, dunque, i parametri più significativi, in modo da renderne più semplice la consultazione. Sarà comunque fornita un'indicazione sugli altri parametri del test.

5.3.1 Variazione della finestra temporale

In questo paragrafo sono riportati i risultati dei test ottenuti al variare della finestra temporale, i.e. il numero di osservazioni passate relative a una metrica. Questo parametro determina, in parte, la topologia della rete e, più precisamente, il numero di nodi dello strato di input. Per quanto riguarda le topologie più complesse, composte da più livelli, verrà fornita l'indicazione sulla dimensione degli strati più interni.

I risultati ottenuti vengono mostrati in tabelle, suddivise in sezioni, ciascuna relativa a una precisa metrica del database, che riportano, oltre alla specifica configurazione del test effettuato, le misure di accuratezza ottenute. Per ogni metrica, i risultati sono ordinati

in modo crescente sulla base del valore assunto dal MAE, in modo da avere, per primo, il *setting* che ha prodotto il risultato migliore e via discorrendo. Nei casi più significativi, vengono inoltre mostrate, in veste grafica, le predizioni generate dalla rete, utile per farsi un'idea più diretta della qualità del modello.

Rete neurale a singolo strato

Innanzitutto, viene presa in esame la topologia più semplice realizzata, la rete neurale a singolo strato.

In questo caso, i test sono stati eseguiti tenendo costanti i parametri indicati in tabella 5.1, secondo i valori riportati.

Tabella 5.1: Parametri invariati per la rete neurale SLP

Orizzonte temporale	1
Funzione di costo	Huber
Ottimizzatore	Adam
Tasso di apprendimento	$1e{-3}$
Dimensione del batch	32
Numero di epoche	300

In tabella 5.2 vengono mostrati i risultati raggiunti utilizzando la topologia a singolo strato.

Tabella 5.2: Risultati ottenuti dalla rete neurale a singolo strato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	6	81,29	193,26	24,44	0,33
	1	81,46	192,44	24,9	0,33
	12	81,96	194,94	24,31	0,34
	32	82,91	195,83	24,49	0,34
	64	83,7	195,44	24,96	0,34
	72	84,47	196,94	24,75	0,35
	144	87,29	200	26,05	0,36
Service CPU Time (jde910)	144	2 597,45	17 647,16	366,81	1,08
	1	2 816,87	17 178,76	167,05	1,17
	6	2 823,59	17 200,89	167,46	1,17
	12	2 831,84	17 218,3	169,26	1,18

Tabella 5.2: Risultati ottenuti dalla rete neurale a singolo strato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (jde910)	32	2 845,23	17 260,73	178,2	1,18
	64	2 855,86	17 329,53	194,46	1,19
	72	2 871,56	17 352,32	200,75	1,19
Service CPU Time (SYS$USERS)	144	7 379,45	12 886,24	91,98	0,58
	64	8 801,46	16 187,02	116,38	0,69
	72	8 824,42	16 146,97	118,41	0,69
	32	8 969,43	16 552,48	120,76	0,7
	6	8 997,6	16 797,98	116,74	0,7
	12	9 007,53	16 631,4	120,53	0,7
	1	9 132,56	16 901,3	122,82	0,71
Database CPU Time	144	8,86	11,34	21,06	0,71
	72	10,45	13,15	25,53	0,84
	64	10,62	13,32	25,94	0,86
	32	10,78	13,66	26,33	0,87
	12	10,99	13,95	27,04	0,89
	6	11,21	14,28	27,68	0,9
	1	11,37	14,46	28,01	0,92
CPU Usage	144	18,63	24,85	38,42	0,88
	72	20,44	28,14	39,92	0,96
	64	20,46	28,2	39,94	0,96
	32	21,26	29,73	40,57	1
	12	21,56	30,18	41,32	1,01
	6	21,75	30,43	43	1,02
	1	22,1	31,38	43,75	1,04
Database Time	144	48,15	71,38	52,6	0,81
	32	51,36	78,7	53,56	0,86
	12	51,47	78,89	53,99	0,86
	72	51,61	78,95	53,18	0,86
	6	51,65	78,96	55,12	0,86
	64	51,69	79,17	53,22	0,86
	1	54,77	83,11	59,18	0,92
Host CPU Utilization	144	1,06	1,43	9,41	0,79
	72	1,15	1,53	10,28	0,85

Tabella 5.2: Risultati ottenuti dalla rete neurale a singolo strato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Host CPU Utilization	64	1,16	1,53	10,43	0,86
	32	1,18	1,56	10,57	0,87
	12	1,21	1,59	11	0,9
	6	1,23	1,62	11,24	0,92
	1	1,39	1,8	12,92	1,04

In figura 5.1 è tracciato l'andamento effettivo della metrica *Service CPU Time (csdb91)*, mettendo a confronto diretto i valori generati dal modello, ossia le predizioni, durante la fase di validazione [1]. Nella fattispecie, è stata considerata la configurazione corrispondente al miglior risultato, ottenuto con una finestra temporale pari a 6 campioni. Come detto in precedenza, dall'osservazione del trend delle due curve è possibile visualizzare in modo più diretto quanto la predizione si discosta dalla realtà.

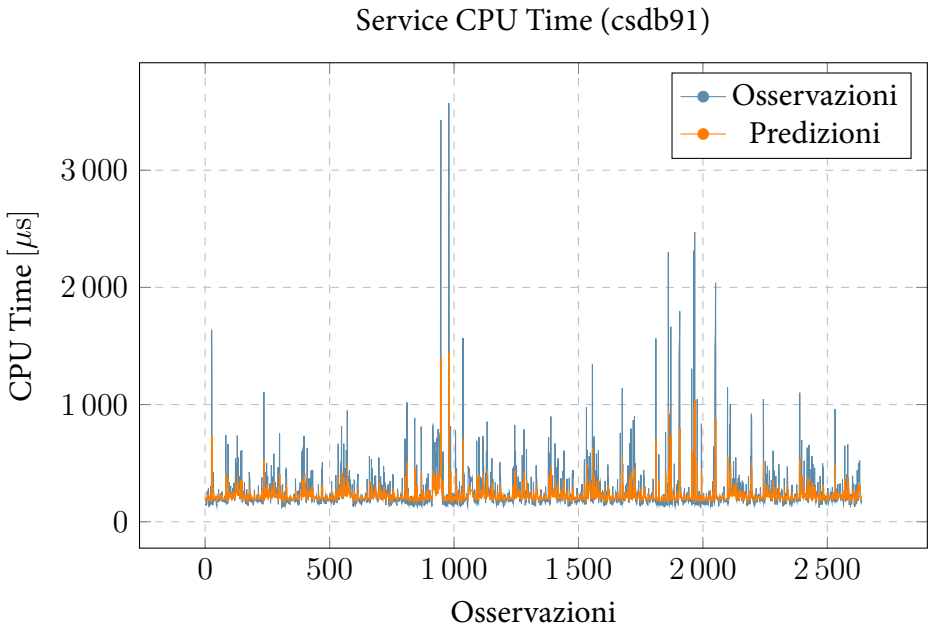


Figura 5.1: Confronto tra osservazioni reali e predizioni riferite alla metrica *Service CPU Time (csdb91)*

[1] I campioni valutati fanno parte dell'insieme di test, in quanto la validazione del modello è stata effettuata su quel set.

È bene precisare che l'obbiettivo dell'analisi, in relazione allo studio svolto, consiste nel realizzare delle proiezioni sull'evoluzione futura delle metriche relative all'istanza del database in esame. Quindi, una volta conclusa la fase di addestramento e validazione, si è passati all'applicazione del modello, come avverrebbe proprio nel contesto reale.

Si consideri la metrica *Service CPU Time (csdb91)* visualizzata in precedenza. La sua proiezione è riportata nel grafico di figura 5.2, dove il numero di osservazioni è stato limitato per mettere in evidenza i valori futuri. Per tracciare l'andamento futuro della metrica è stato utilizzato, nello specifico, l'approccio *ricorsivo*, discusso nel paragrafo 4.3.5: le ultime 6 osservazioni della serie sono state elaborate dal modello, ottenendo così il valore all'istante di tempo successivo (10 minuti). Questo processo è stato iterato più volte, utilizzando il valore predetto come ultima osservazione.

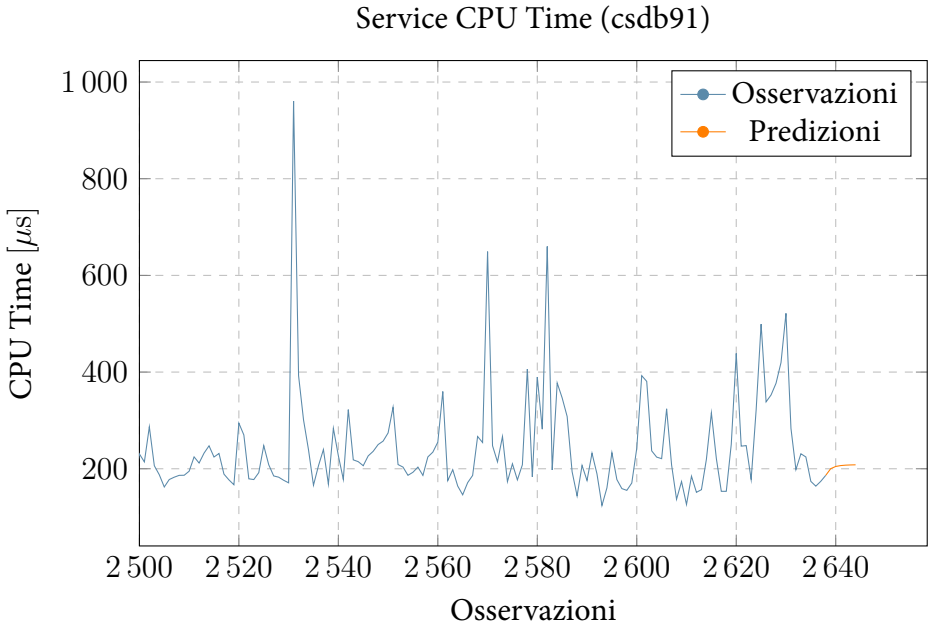


Figura 5.2: Predizione della metrica *Service CPU Time (csdb91)* per un orizzonte temporale pari a 6 campioni

Come si può osservare dai risultati in tabella, all'aumentare della finestra corrisponde, nella maggior parte dei casi, un miglioramento nella qualità delle predizioni. Questo è un risultato intuibile, in quanto, dovendo predire l'istante di tempo successivo, basarsi su un arco temporale più o meno esteso influisce sul training: il modello dispone di un numero di informazioni maggiore con cui predire il prossimo valore. D'altra parte, non è detto che ciò avvenga per ogni metrica, come si può notare dai risultati in tabella 5.2; inoltre, non è banale capire fino a che punto l'aumento della finestra temporale comporta un miglioramento nella predizione.

Rete neurale multistrato

In questo paragrafo vengono analizzati i risultati ottenuti tramite la rete neurale multistrato, considerando sempre come variabile la finestra temporale. Come nel caso precedente, sono stati tenuti costanti gli altri parametri del modello (si veda la tabella 5.3).

Tabella 5.3: Parametri invariati per la rete neurale MLP

Dim. strato nascosto n. 1	32
Dime. strato nascosto n. 2	32
Orizzonte temporale	1
Funzione di costo	Huber
Ottimizzatore	Adam
Tasso di apprendimento	$1e{-3}$
Dimensione del batch	32
Numero di epoche	100

Come si può notare, il numero di epoche non corrisponde rispetto al caso analizzato in precedenza. È stato scelto un numero inferiore in quanto si è notato che la rete multistrato tende a divergere per un numero di iterazioni maggiore, rispetto alla rete a singolo strato. Inoltre, sono presenti due parametri aggiuntivi: la dimensione del primo strato nascosto e del secondo, ovvero il numero di neuroni che compongono i due livelli, parametro che, per ovvie ragioni, non è presente nella rete a singolo strato.

Tabella 5.4: Risultati ottenuti dalla rete neurale multistrato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	6	80,6	197,12	24,03	0,33
	12	82,67	200,95	24,57	0,34
	1	83,47	197,38	25,8	0,34
	32	84,57	199,29	26,24	0,35
	72	85,53	205,26	26,61	0,35
	64	85,7	207,78	26,35	0,35
	144	88,38	203,2	28,26	0,36
Service CPU Time (jde910)	144	2 591,72	18 125,01	302,81	1,08
	12	2 764,53	19 456,24	155,85	1,15
	32	2 820,29	21 183,72	221,24	1,17
	1	2 831,84	17 325,21	198,62	1,17

Tabella 5.4: Risultati ottenuti dalla rete neurale multistrato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (jde910)	72	2 887,02	19 851,91	256,21	1,2
	64	2 913,52	20 114,16	227,98	1,21
	6	3 009,1	23 321,8	238,16	1,25
Service CPU Time (SYS$USERS)	64	7 845,19	13 683,52	112,79	0,61
	144	8 480,07	14 999,43	113,11	0,66
	12	8 629,24	15 666,75	119,36	0,67
	72	8 697,85	15 184,85	118,29	0,68
	6	8 710,94	16 360,03	115,85	0,68
	32	8 776,86	15 684,84	125,05	0,68
	1	9 045,3	17 079,82	114,85	0,71
Database CPU Time	32	10,62	13,43	24,69	0,86
	12	10,63	13,48	25,56	0,86
	64	10,66	13,41	24,28	0,86
	72	10,92	13,68	24,82	0,88
	6	10,96	13,98	26,67	0,88
	144	11,28	14,26	26,4	0,91
	1	11,33	14,41	27,84	0,91
CPU Usage	64	19,47	26,68	36,43	0,91
	72	19,65	27,05	36,5	0,92
	144	19,69	26,82	37,39	0,93
	32	19,82	27,21	38,22	0,93
	12	20,24	28,19	38,2	0,95
	6	20,84	29,15	38,92	0,98
	1	21,83	30,56	44,34	1,03
Database Time	144	39,87	56,66	39,83	0,67
	64	40,95	58	41,44	0,68
	72	41,62	59,78	41,02	0,7
	32	42,46	60,45	43,26	0,71
	12	46,29	72,76	45,99	0,77
	6	49,8	76,78	51,99	0,83
	1	53,71	81,18	56,89	0,9
Host CPU Utilization	144	1,15	1,54	10,01	0,85
	64	1,19	1,59	10,47	0,88

Tabella 5.4: Risultati ottenuti dalla rete neurale multistrato al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Host CPU Utilization	12	1,2	1,6	10,7	0,9
	72	1,22	1,61	10,77	0,91
	6	1,22	1,62	10,95	0,91
	32	1,27	1,69	11,15	0,95
	1	1,36	1,77	12,49	1,01

Al variare della finestra temporale, la rete MLP presenta grosso modo lo stesso comportamento della rete SLP: a sequenze di campioni più lunghe corrispondono generalmente prestazioni migliori, in termini di accuratezza. Allo stesso modo, sono presenti delle eccezioni, come risulta dall'osservazione dei risultati relativi alla metrica *Service CPU Time (csdb91)*.

Un risultato migliore, rispetto al modello analizzato precedentemente, è stato ottenuto per la metrica *Database Time*, di cui viene mostrato il confronto tra valori reali e predetti in figura 5.3.

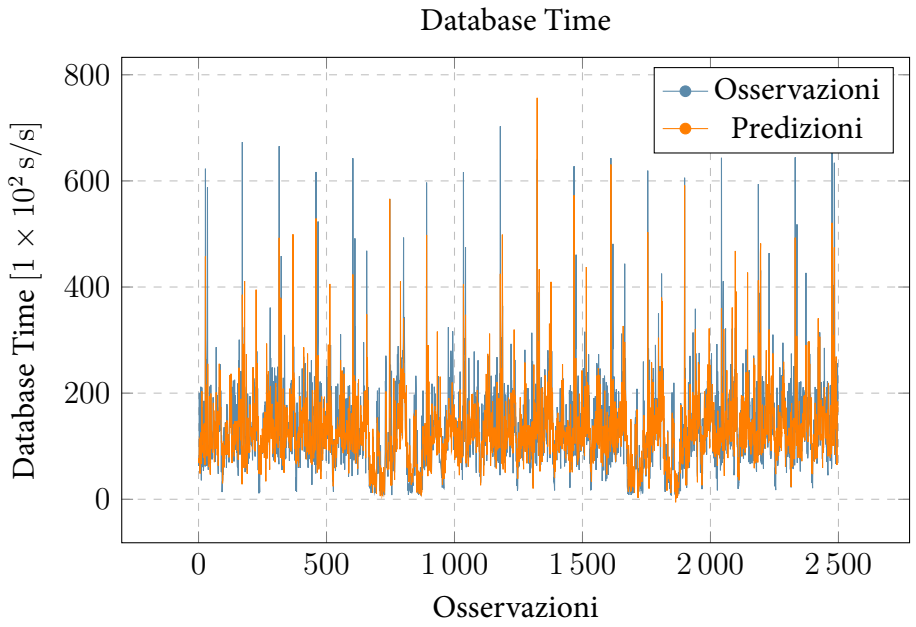


Figura 5.3: Confronto tra osservazioni reali e predizioni riferite alla metrica *Database Time*

Rete LSTM

Restano da esaminare i risultati dei test eseguiti tramite la rete neurale LSTM. Come in precedenza, la tabella 5.5 riporta i valori dei parametri mantenuti invariati. Nel caso di una rete LSTM bisogna specificare il numero di layer nascosti che compongono la rete, oltre alla loro dimensione.

Tabella 5.5: Parametri invariati per la rete neurale LSTM

Numero di layer nascosti	2
Dimensione strato nascosto	32
Orizzonte temporale	1
Funzione di costo	Huber
Ottimizzatore	Adam
Tasso di apprendimento	$1e{-3}$
Dimensione del batch	32
Numero di epoche	30

Tabella 5.6: Risultati ottenuti dalla rete neurale LSTM al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	32	77,77	191,33	22,98	0,23
	72	78,48	193,34	22,95	0,23
	64	78,51	194,76	23,21	0,23
	72	78,51	192,66	23,37	0,23
	6	79,14	194,07	23,34	0,23
	144	79,2	198,27	23,07	0,23
	1	83,21	198,71	26,04	0,24
Service CPU Time (jde910)	64	2 877,82	17 364,27	42,12	1,44
	72	2 887,49	17 396,16	40,99	1,44
	12	2 889,81	17 252,45	41,71	1,44
	6	2 899,37	17 270,36	34,82	1,45
	1	2 914,62	17 278,93	37,03	1,45
	144	2 923,33	17 613,74	34,75	1,46
	32	2 925,3	17 364,28	29,9	1,46
Service CPU Time (SYS$USERS)	144	8 021,54	14 182,35	111,1	0,92
	64	8 035,98	14 326,86	111,17	0,92

Tabella 5.6: Risultati ottenuti dalla rete neurale LSTM al variare della finestra temporale

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (SYS$USERS)	32	8 125,95	14 708,67	111,19	0,93
	72	8 282,33	14 780,29	113,25	0,95
	12	8 762,95	16 345,93	113,43	1,01
	6	8 981,32	16 765,62	128	1,03
	1	9 133,67	17 212,01	118,85	1,05
Database CPU Time	144	10,13	12,84	24,29	0,87
	72	10,14	12,83	24,56	0,87
	32	10,25	13,01	24,8	0,88
	64	10,44	13,2	25,29	0,89
	12	10,6	13,39	25,61	0,91
	6	11,11	14,16	27,32	0,95
	1	11,35	14,4	27,83	0,97
CPU Usage	144	18,93	25,66	38,03	0,76
	72	19	25,74	37,61	0,76
	32	19,12	26,29	36,6	0,76
	64	19,56	26,96	37	0,78
	12	19,7	27,24	37,35	0,79
	6	20,49	28,36	40,29	0,82
	1	21,93	30,73	44,37	0,88
Database Time	144	48,18	73,09	47,96	0,7
	72	48,46	72,73	48,1	0,71
	64	48,5	74,99	46,54	0,71
	12	47,06	72,94	47,48	0,69
	32	47,77	73,43	47,74	0,7
	6	49,66	77,11	49,07	0,72
	1	53,2	80,61	56,39	0,77
Host CPU Utilization	6	1,2	1,6	10,69	0,6
	32	1,21	1,61	10,71	0,6
	12	1,21	1,61	10,78	0,61
	72	1,21	1,61	10,79	0,61
	64	1,21	1,61	10,79	0,61
	144	1,22	1,62	10,82	0,61
	1	1,34	1,73	12,35	0,67

Contrariamente a quanto ci si potesse aspettare, aumentando la complessità della rete, i risultati non migliorano in maniera consistente, anzi, in taluni casi si sono riscontrati dei valori di accuratezza persino peggiori (in riferimento alla configurazione considerata).

Rispetto ai precedenti casi, è stato ottenuto un valore di accuratezza migliore per la metrica *Service CPU Time (csdb91)*. In figura 5.4 vengono riportati i valori reali e le predizioni relative, in modo da poter anche fare un confronto diretto con il grafico in figura 5.1, relativo alla stessa metrica.

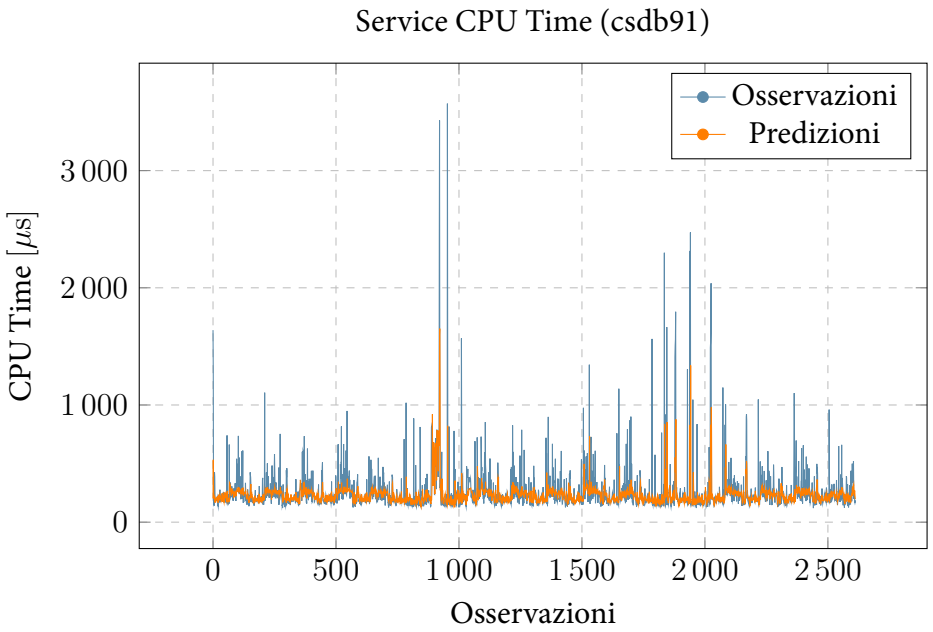


Figura 5.4: Confronto tra osservazioni reali e predizioni riferite alla metrica *Service CPU Time (csdb91)*

Il forecast dei successivi 6 campioni (corrispondenti a 1 ora) viene, invece, mostrato in figura 5.5.

5.3.2 Variazione dell'orizzonte temporale

Si considerano, in questa sezione, i valori ottenuti, relativamente alla metrica *Service CPU Time (csdb91)*, impostando un orizzonte temporale maggiore di 1. Nello specifico, sono stati considerati 3 diversi orizzonti temporali (pari, rispettivamente, a un'ora, due ore e più di cinque ore), in relazione a 2 finestre temporali, corrispondenti a un giorno e due giorni.

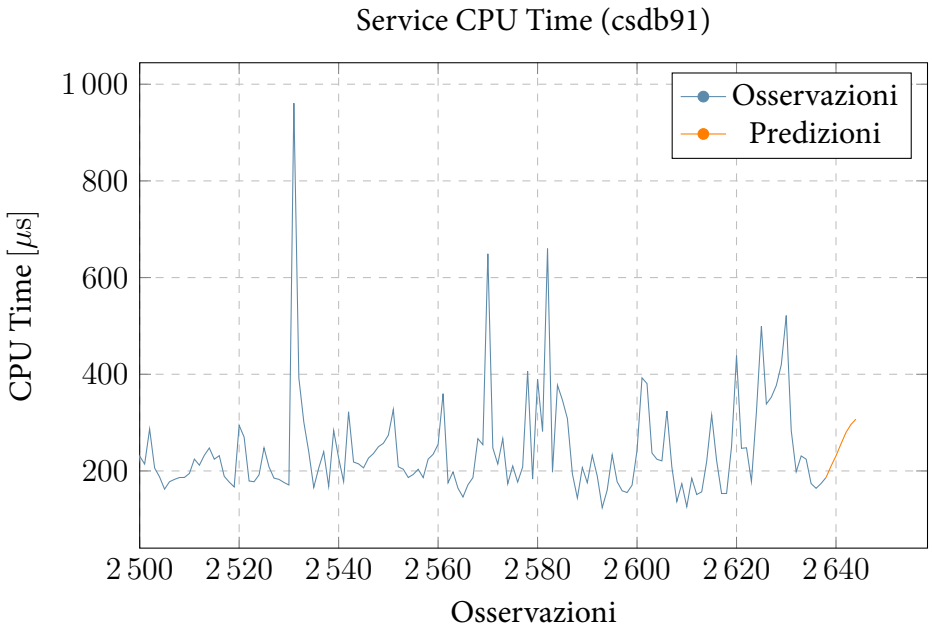


Figura 5.5: Predizione della metrica *Service CPU Time (csdb91)* per un orizzonte temporale pari a 6 campioni

Rete neurale a singolo strato

La tabella 5.7 si riferisce ai risultati raggiunti dalla rete neurale a singolo strato. I valori degli altri parametri sono gli stessi indicati nella tabella 5.1 a pagina 62, eccetto, ovviamente, per l'orizzonte temporale.

Tabella 5.7: Risultati ottenuti dalla rete neurale a singolo strato, al variare dell'orizzonte e della finestra temporale

Metrica	Finestra temp.	Orizz. temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	144	12	92,15	217,17	26,03	0,27
	144	32	92,42	216,85	26,04	0,27
	144	6	92,85	218,26	27,28	0,27
	288	12	97,11	224,92	27,24	0,28
	288	32	97,49	224,7	27,79	0,28
	288	6	98,44	225,35	28,85	0,29

Viene mostrato, inoltre, il grafico, in figura 5.6, che presenta la proiezione nel futuro delle due ore successive, i.e. 12 punti, utilizzando la configurazione corrispondente al risultato più accurato.

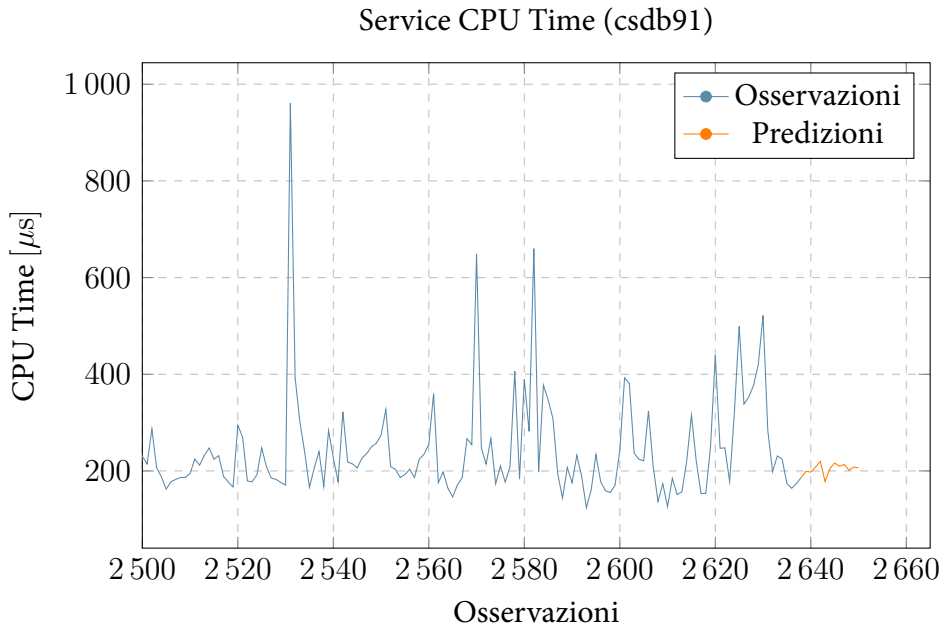


Figura 5.6: Predizione della metrica *Service CPU Time (csdb91)* per un orizzonte temporale pari a 12 campioni

Rete neurale multistrato

In questa sezione sono riportati i valori di accuratezza ottenuti per la rete neurale multistrato. Nei test seguenti, è stato utilizzato un numero di epoche pari a 50 e un batch contenente 72 sequenze. Si può fare riferimento alla tabella 5.3 a pagina 66 per i valori dei parametri restanti.

Tabella 5.8: Risultati ottenuti dalla rete neurale multistrato al variare dell'orizzonte e della finestra temporale

Metrica	Finestra temp.	Orizz. temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	144	12	89,5	208,91	26,72	0,26
	288	12	90,68	214,54	26,08	0,27
	144	32	91,56	215,47	26,62	0,27
	144	6	93,17	216,28	28,57	0,27
	288	6	97,13	216,87	30,9	0,28
	288	32	97,53	220,38	29,5	0,29

Anche per quanto riguarda la rete neurale MLP viene riportato il grafico (figura 5.7) relativo alla predizione delle due ore seguenti.

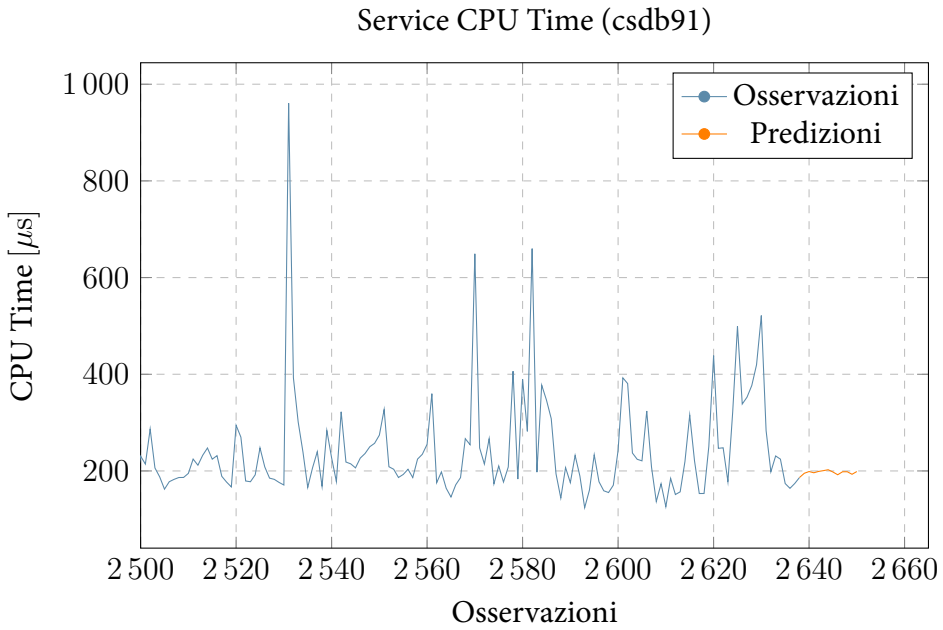


Figura 5.7: Predizione della metrica *Service CPU Time (csdb91)* per un orizzonte temporale pari a 12 campioni

Rete LSTM

Infine, come si può leggere dalla tabella 5.9, la rete LSTM ha generato il risultato migliore per un orizzonte temporale pari a 6 e il forecast della metrica è riportato in figura 5.8. Il numero di epoche è 20, in tal caso, mentre la dimensione del batch è pari a 128. Si faccia riferimento alla tabella 5.5 a pagina 69 per gli altri parametri.

Tabella 5.9: Risultati ottenuti dalla rete neurale LSTM al variare dell'orizzonte e della finestra temporale

Metrica	Finestra temp.	Orizz. temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	144	6	85,68	209,52	24,25	0,25
	144	12	85,73	210	24,06	0,25
	288	12	86,75	214,46	23,94	0,25
	288	6	87,09	213,21	24,73	0,25
	144	32	88,7	214,05	24,81	0,26
	288	32	88,85	217,31	24,57	0,26

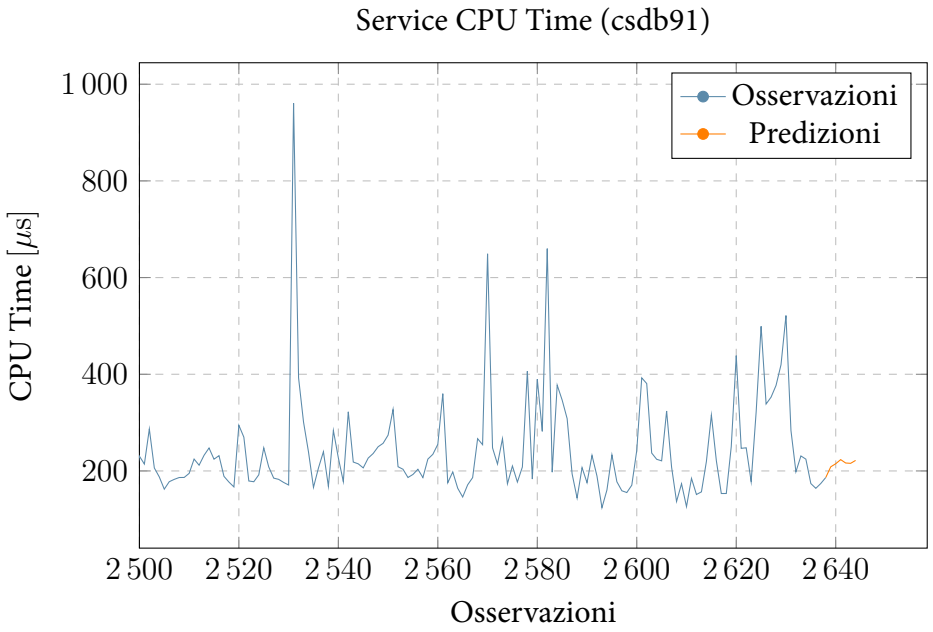


Figura 5.8: Predizione della metrica *Service CPU Time (csdb91)* per un orizzonte temporale pari a 6 campioni

5.3.3 Funzioni di costo e ottimizzatore

Per quanto riguarda le funzioni di costo e il metodo di ottimizzazione utilizzati nell'addestramento delle reti neurali, si è riscontrato, nella maggior parte dei casi, che la funzione di costo Huber e, in certi casi, l'errore medio assoluto (MAE), in unione al metodo di ottimizzazione Adam, forniscono delle prestazioni tendenzialmente migliori, rispetto all'applicazione della funzione di costo errore quadratico medio (MSE) e al metodo di discesa stocastica del gradiente (SGD).

A titolo di esempio, in tabella 5.10, vengono riportati i risultati ottenuti tramite la rete neurale a singolo strato corrispondenti alla metrica *Database Time*. I valori dei parametri non specificati sono quelli indicati nella tabella 5.1 a pagina 62.

Tabella 5.10: Risultati ottenuti dalla rete neurale SLP al variare della funzione di costo e del metodo di ottimizzazione

Metrica	Finestra temp.	Funzione di costo	Ottim.	MAE	RMSE	MAPE	MASE
Database Time	64	mae	adam	51,09	79,43	46,45	0,74
	64	huber	adam	51,69	79,17	53,22	0,75
	64	huber	sgd	52,08	79,36	54,38	0,76

Tabella 5.10: Risultati ottenuti dalla rete neurale SLP al variare della funzione di costo e del metodo di ottimizzazione

Metrica	Finestra temp.	Funzione di costo	Ottim.	MAE	RMSE	MAPE	MASE
Database Time	64	mae	sgd	52,17	80,37	48,45	0,76
	64	mse	adam	60,35	87,11	78,17	0,88
	64	mse	sgd				

In particolare, l'accoppiata MSE e SGD, per alcune metriche, non produce alcun risultato, ovvero, non riesce a convergere (in tabella 5.10 viene mostrato un esempio in relazione alla metrica *Database Time*). In questi casi è necessario calibrare il tasso di apprendimento, riducendolo, in modo da evitare il fenomeno dell'esplosione del gradiente, come evidenziato in figura 5.9. Le considerazioni appena fatte valgono per tutte le reti prese in esame.

5.3.4 Variazione del numero di attributi

Per la predizione di una metrica, finora sono state considerate esclusivamente serie univariate. Al fine di comprendere se il contributo di più metriche possa aiutare l'algoritmo a riconoscere dei pattern significativi, sono stati condotti degli esperimenti anche su serie multivariate. In tabella 5.11 sono presenti le misure di accuratezza riferite alla

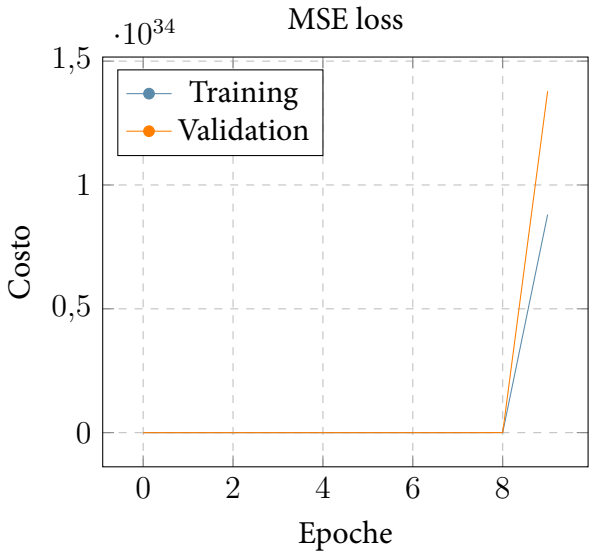


Figura 5.9: *Training* e *validation loss* relativi alla metrica *Database Time*

metrica *CPU Usage*, ottenute mediante una rete neurale LSTM. Nel caso testato, numero di epoche e dimensione del batch sono pari, rispettivamente, a 50 e 72.

Tabella 5.11: Risultati ottenuti tramite la rete LSTM al variare del numero di attributi

Metrica	Attributi	Finestra temp.	MAE	RMSE	MAPE
CPU Usage	CPU Usage Host CPU Utilization	144	19,6	26,36	38,63
CPU Usage	Service CPU Time (csdb91) Service CPU Time (jde910) Service CPU Time (SYS$USERS) CPU Usage	144	22,82	31,06	45,27
CPU Usage	Average Instance CPU Consistent Read Gets CPU Usage Session Logical Reads	144	19,37	26,4	38,44

Considerare un numero di attributi maggiore, facendo particolare riferimento ai risultati raccolti, non ha portato a dei miglioramenti significativi, almeno per quanto concerne le configurazioni testate. Riguardo le metriche di accuratezza, nel caso delle serie multivariate, non è stato calcolato il MASE.

I test sono stati eseguiti tenendo conto sia di attributi con indice di correlazione di Pearson relativamente basso (minore di 0.5), come nei primi due casi riportati in tabella; sia attributi con un indice di correlazione più significativo (maggiore di 0.5), come nell'ultimo caso. È possibile notare un modesto miglioramento se si considerano attributi caratterizzati da una correlazione più forte.

5.3.5 Trasformazione dei dati di input

Vengono mostrati, in questa sezione, i casi in cui l'insieme di dati viene sottoposto al modello dopo opportune trasformazioni, come la differenziazione o la rimozione degli outlier.

Si può facilmente intuire come la rimozione delle anomalie riesca a garantire dei risultati tendenzialmente più accurati, o quantomeno paragonabili a quelli ottenuti altrimenti. In altre parole, si attua una 'semplificazione' del problema, che permette alla

rete di imparare in modo più accurato la relazione che sussiste tra valori di ingresso e di uscita.

D'altra parte, differenziando i dati a disposizione, si cerca di addestrare la rete non tanto sui valori assunti dalle metriche nel tempo, quanto sulle differenze, calcolate nel tempo, che caratterizzano le osservazioni di una serie. Questo tipo di operazione, molto utilizzata in approcci tradizionali, non è detto che fornisca dei miglioramenti sostanziali per quanto riguarda il contesto in esame.

In tabella 5.12 sono riportati le misure di accuratezza ottenute tramite l'applicazione della rete neurale MLP alle metriche indicate (per un numero di epoche pari a 50 e una dimensione del batch di 72), dopo aver eliminato, con il metodo interquartile, i valori 'anomali'. Soprattutto per quelle metriche che presentano dei picchi considerevoli, sono stati registrati dei netti miglioramenti.

Tabella 5.12: Risultati ottenuti dalla rete neurale multistrato in seguito alla rimozione degli *outlier*

Metrica	Finestra temp.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	144	47,5	64,34	21,93	0,89
Service CPU Time (jde910)	144	6,13	8,6	3,94	0,61
Service CPU Time (SYS$USERS)	144	4 981,59	7 142,78	99,2	0,84
Database CPU Time	144	10,33	13	23,94	0,88
CPU Usage	144	20,1	26,24	41,92	0,69
Database Time	144	44,68	57,82	48,48	0,7
Host CPU Utilization	144	1,06	1,43	9,44	0,53

Per quanto concerne le serie differenziate, sono stati condotti alcuni test, per ogni topologia di rete, senza tuttavia ottenere dei risultati particolarmente soddisfacenti. Comunque, come riportato nella tabella 5.15, nel paragrafo 5.3.7, dove sono elencati i migliori risultati generati dalla rete LSTM, si può notare che per alcune metriche questo tipo di trasformazione ha contribuito a migliorare l'accuratezza della predizione.

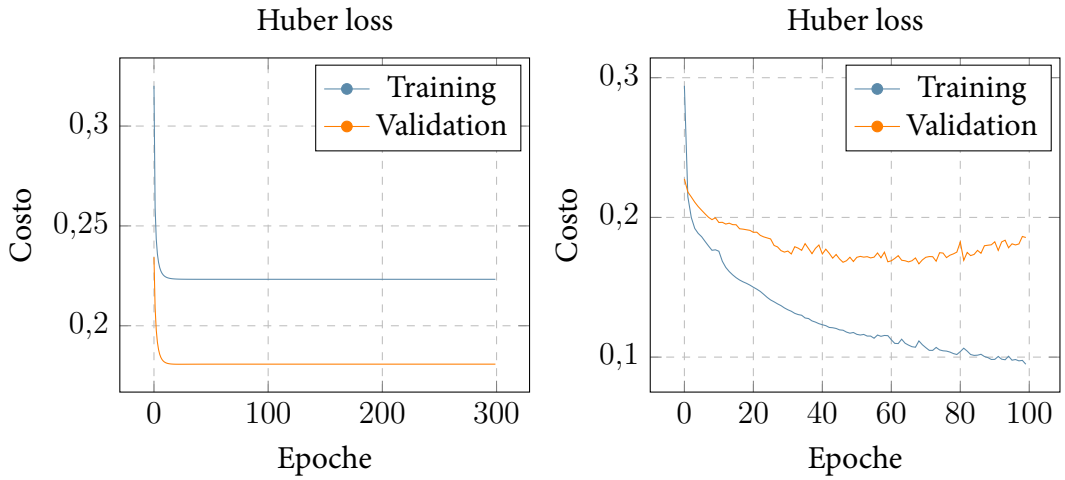
Un'altra trasformazione applicata ai dati di ingresso, prima di essere elaborati dal modello, è lo scaling, di cui si è discusso nel paragrafo 4.3.1. In tal caso, la maggior parte degli esperimenti sono stati eseguiti su dati scalati secondo la trasformazione effettuata dalla classe `RobustScaler`. Non sono stati notati particolari cambiamenti effettuando un'operazione di standardizzazione sui dati tramite la classe `StandardScaler`.

5.3.6 Altri parametri

Gli altri parametri valutati riguardano il numero di epoche, il tasso di apprendimento e la dimensione del batch.

Una rete neurale raggiunge la convergenza dopo aver indagato le possibili soluzioni al problema fino a individuare un punto di minimo (locale o globale). In questo processo, i parametri sopra citati rivestono un ruolo molto importante, influenzando sia la capacità di convergenza della rete che i tempi in cui viene raggiunta. In particolare, il numero di epoche stabilisce quante iterazioni effettuare perché la rete converga; il tasso di apprendimento determina l'entità della variazione verso la soluzione; infine, la dimensione del batch indica il numero di istanze del problema che la rete valuta assieme.

Nei vari test, il numero di epoche è stato impostato in base alla complessità della rete e del problema: per topologie di reti più complesse, si è notato, che un numero di iterazioni troppo elevato potrebbe portare la rete a divergere. Si osservino, a tal riguardo, i grafici di figura 5.10 che riportano l'andamento del *training* e *validation loss* corrispondenti all'addestramento di una rete SLP (figura 5.10(a)) e di una rete LSTM (figura 5.10(b)) in relazione alla metrica *CPU Usage*. Come si evince dall'osservazione delle curva, la rete a singolo strato si stabilizza dopo un certo numero di epoche, mentre la rete LSTM tende a divergere.



(a) Rete SLP

(b) Rete LSTM

Figura 5.10: Andamento del *training loss* e *validation loss* relativi all'addestramento di due reti neurali, SLP e LSTM, sulla metrica *CPU Usage*

Il tasso di apprendimento determina quanto cambino i pesi della rete a ogni passo dell'algoritmo. Quindi, per valori troppo grandi la rete potrebbe non riuscire a convergere; mentre, per valori troppo piccoli, i tempi di convergenza aumentano considerevol-

mente. Nella maggior parte delle sperimentazioni, è stato adottato un valore di compromesso, ovvero $1e-3$. Sono stati condotti alcuni test variando il tasso di apprendimento di uno o due ordini di grandezza, senza riscontrare particolare miglioramenti. Inoltre, avendo utilizzato, nella maggior parte dei casi, un metodo di ottimizzazione adattativo (Adam), la calibrazione di tale parametro non è stata particolarmente esplorata.

Infine, la dimensione del batch impatta sia sulle prestazioni della rete che sui tempi di training: maggiore è la dimensione del batch più sono le istanze del problema che la rete valuta contemporaneamente. Pertanto, la correzione dell'errore avviene su più istanze e ciò influisce sia sulla convergenza che, ovviamente, sulla velocità di addestramento della rete.

Sono state valutate dimensioni di batch differenti e, nel caso specifico delle reti più complesse, sono stati presi in considerazione batch più grandi, per accelerare la fase di training del modello. Dai test effettuati, si sono ottenuti risultati migliori per batch di dimensioni relativamente ridotte (inferiori a 300 istanze).

Nel paragrafo 5.3.7, dove sono riportate le configurazioni che hanno generato i risultati migliori, si può fare riferimento ai parametri appena descritti, per avere un'idea più precisa sui valori corrispondenti.

5.3.7 Risultati migliori

Le tabelle 5.13, 5.14 e 5.15, riportate nelle pagine seguenti, mostrano, per ogni rete neurale implementata, le configurazioni (i.e., parametri della rete, algoritmi utilizzati eccetera) per cui si sono raggiunti i risultati migliori in relazione alle metriche esaminate, permettendo così un confronto diretto tra le prestazioni dei diversi modelli. Per ogni riga viene riportata la metrica analizzata e i valori relativi a ciascun parametro. Le metriche considerate non sono state ripulite dagli outlier.

Paragonando le misure di accuratezza ottenute, si evince che la rete multistrato fornisce, mediamente, le prestazioni migliori.

Tabella 5.13: Risultati migliori ottenuti attraverso la rete neurale SLP e relativa configurazione

Metrica	Fin. temp.	Orizz. temp.	Funz. costo	Ottim.	Dim. batch	Ep.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	1	1	mae	adam	32	300	80,91	194,53	23,51	0,24
Service CPU Time (jde910)	144	1	huber	adam	32	300	2 597,45	17 647,16	366,81	1,3
Service CPU Time (SYS$USERS)	144	1	huber	sgd	32	300	7 343,95	12 818,32	91,74	0,84
Database CPU Time	144	1	huber	adam	32	300	8,86	11,34	21,06	0,76
CPU Usage	288	1	huber	adam	32	300	17,49	23,66	35,52	0,7
Database Time	288	32	huber	adam	32	300	45,25	63,38	58,69	0,66
Host CPU Utilization	288	1	huber	adam	32	300	1,03	1,38	9,31	0,51

Tabella 5.14: Risultati migliori ottenuti attraverso la rete neurale MLP e relativa configurazione

Metrica	Fin. temp.	Orizz. temp.	Funz. costo	Ottim.	H1	H2	Dim. batch	Ep.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	6	1	mae	adam	32	32	32	100	78,9	197,03	22,13	0,23
Service CPU Time (jde910)	144	1	huber	adam	32	32	128	50	2 379,27	16 791,85	161,83	1,19
Service CPU Time (SYS$USERS)	144	1	huber	sgd	32	16	72	50	7 012,74	12 665,2	92,31	0,81
Database CPU Time	288	6	huber	adam	32	32	72	50	9,52	12,06	21,58	0,81
CPU Usage	144	1	huber	adam	64	32	72	50	17,01	23,64	32,35	0,68
Database Time	144	1	huber	adam	64	32	72	50	36,81	51,4	39,39	0,54
Host CPU Utilization	144	1	mse	adam	32	32	72	50	1,04	1,39	9,22	0,52

Tabella 5.15: Risultati migliori ottenuti attraverso la rete neurale LSTM e relativa configurazione

Metrica	Fin. temp.	Orizz. temp.	Diff.	Funz. costo	Ottim.	H1	Num. layer	Dim. batch	Ep.	MAE	RMSE	MAPE	MASE
Service CPU Time (csdb91)	144	1	0	huber	adam	32	1	128	50	77,27	190,65	23,17	0,23
Service CPU Time (jde910)	72	1	0	huber	adam	32	2	32	50	2 857,34	17 324,35	52,83	1,43
Service CPU Time (SYS$USERS)	72	1	0	huber	adam	24	1	128	50	7 917,14	14 220,11	107,89	0,91
Database CPU Time	72	1	0	huber	adam	32	2	32	50	10	12,75	23,61	0,86
CPU Usage	144	1	144	huber	adam	32	1	72	50	18,8	27,47	39,55	0,76
Database Time	144	1	144	huber	adam	32	1	72	50	36,9	56,74	42,2	0,54
Host CPU Utilization	32	1	0	mae	adam	32	2	32	50	1,17	1,57	10,34	0,58

Capitolo 6

Conclusioni

6.1 Considerazioni di carattere generale

Argomento di quest'ultimo capitolo è la disamina dei risultati ottenuti dall'esecuzione dei vari test, che si traduce nella valutazione critica dei diversi aspetti che hanno portato a tali risultati.

In particolare, si cercherà di fornire un'interpretazione più incentrata sull'aspetto pratico, cercando, dunque, di calare l'analisi svolta all'interno del contesto preso in esame. Finora, difatti, il problema è stato affrontato con il fine mirato di migliorare il più possibile l'accuratezza dei diversi modelli realizzati. In altri termini, i test eseguiti, secondo le diverse configurazioni, sono serviti per determinare gli algoritmi e i parametri tali da generare il miglior risultato possibile che, nell'ottica di questo lavoro, si esprime nella capacità della rete di tracciare l'andamento nel tempo delle metriche del database.

Inoltre, si intendono valutare anche gli aspetti più critici sia in riferimento al problema esaminato che a un contesto più generico: quei casi in cui un'analisi del genere potrebbe risultare di difficile attuazione o poco utile o, nel caso peggiore, portare a conclusioni errate. Gli aspetti critici della soluzione proposta saranno argomento del paragrafo 6.3 a pagina 87.

Il lavoro di ricerca si è concentrato soprattutto sulla parte di *forecast* delle metriche di un sistema RDBMS, interpretate come serie di dati, o, per essere più precisi, come variabili casuali ordinate nel tempo (i.e., serie temporali). A tal proposito, sono stati definiti alcuni modelli, basati su algoritmi di apprendimento automatico, per elaborare le serie di dati (i.e., le metriche), quindi per trarne i valori futuri. Tuttavia, il lavoro svolto non ha assolutamente la pretesa di presentarsi come un lavoro conclusivo, bensì sono numerosi gli aspetti che possono ancora essere indagati, come, per esempio, il tipo di analisi condotta sui dati o le innumerevoli possibilità implementative dei modelli di reti neurali. Pertanto nel paragrafo 6.4 si prenderanno in considerazione alcuni dei possibili

sviluppi che possano contribuire a rendere lo strumento di analisi realizzato più efficace.

6.2 Valutazione dei risultati

L'interpretazione dei risultati ottenuti dall'esecuzione dei vari test non può esimersi dal considerare il contesto specifico analizzato. Come più volte è stato ripetuto nel corso di questa trattazione, il caso di studio è rappresentato dall'istanza del database di un cliente. Quindi, i dati processati dagli algoritmi fanno riferimento ai parametri che caratterizzano, appunto, un sistema di basi di dati. Per essere più precisi, è stato preso in considerazione solo un sottoinsieme delle metriche che descrivono il sistema e, in particolare, sono state analizzate quelle relative alle performance. Questo aspetto è stato trattato nel capitolo 2, in cui si è affrontato il problema di definire il *problema* (per usare un gioco di parole) in termini di performance del database. Ciò su cui si vuole porre l'attenzione, in questo paragrafo, è valutare il problema in relazione ai risultati ottenuti grazie all'analisi. In altre parole, si vuole esaminare l'attendibilità di un determinato modello in rapporto al contesto per cui viene applicato.

Nel capitolo 5 a pagina 59, in cui sono riportate varie sperimentazioni effettuate su un set di metriche, si è tenuto conto della qualità di ciascun modello sulla base di alcune misure di accuratezza (i.e., MAE, RMSE, MAPE e MASE). La difficoltà risiede proprio nel dare il giusto peso e la corretta interpretazione ai valori espressi dalle misure appena citate. È per questo motivo che non bisogna trascurare il contesto particolare per cui vengono calcolate.

Le metriche di accuratezza utilizzate per valutare le prestazioni dei modelli sono state scelte proprio per fornire una prima interpretazione ai risultati raggiunti e poter trarne delle conclusioni integrando tale dato con le informazioni relative al dominio specifico. Per fare un esempio, l'accuratezza misurata nella valutazione della metrica *Service CPU Time (jde910)* è piuttosto scadente, resa tale per la presenza di valori molto al di sopra della media. Facendo riferimento alle misure riportate nella tabella 5.2 a pagina 64, inerenti alla rete SLP, si osserva un valore di MAE pari a $2597,45$. Questo indice rappresenta lo scarto medio assoluto, ma, nel contesto in esame, ciò indica che il modello commette un errore, in media, di circa 2500 microsecondi, in quanto la metrica in questione rappresenta, come specificato nella documentazione Oracle, il consumo medio di CPU relativamente alle richieste per un particolare servizio (in questo caso, *jde910*). Ragionevolmente, un modello che ammette un errore di 3 ordini di grandezza non può essere considerato attendibile.

Un'analisi di questo tipo richiede, in sintesi, le seguenti considerazioni: definire qual è il comportamento 'normale' di una metrica e stabilire di conseguenza delle soglie per cui considerare il valore ottenuto accettabile o meno, cioè indice di un possibile problema. In teoria, il modello appena considerato potrebbe anche essere ammesso, in quanto errori dell'ordine dei millisecondi, per la metrica in questione, potrebbero comunque

non rappresentare un problema. Tuttavia, un margine di incertezza così ampio potrebbe compromettere la definizione di 'comportamento normale'.

Per rendere più chiaro il concetto, si consideri, per un dato sistema, un consumo di CPU pari all'80%. Si potrebbe indurre a pensare, osservando esclusivamente la percentuale di utilizzo, senza ulteriori informazioni, che tale sistema sia sottoposto a un carico eccessivo e che stia lavorando in condizioni anomale. Se, invece, venisse appurato che la CPU viene utilizzata all'80% nella maggior parte del tempo, allora ciò dovrebbe essere considerato il comportamento standard del sistema, cambiando in tal modo l'interpretazione sul valore assunto dalla metrica.

Dunque, l'interesse principale del lavoro di ricerca è descrivere al meglio lo stato del sistema, in modo da poter proiettare il trend evolutivo e prevedere eventuali cambiamenti anomali nel suo andamento, attraverso il confronto con la storia passata del sistema.

In sintesi, sebbene l'attendibilità di una misura dipenda dalla specifica metrica e conviene che sia valutata da un esperto del dominio, un risultato può essere ritenuto ragionevole se l'errore commesso è ridotto, non maggiore di una decina di punti percentuali, come nel caso della metrica *Host CPU Utilization*, o comunque tale da permettere di riprodurre un pattern significativo e coerente con l'andamento usuale della metrica.

6.3 Aspetti critici

La soluzione proposta non è esente da aspetti che ne possano inficiare il corretto funzionamento. Oltre alle criticità già evidenziate nel paragrafo 6.2, inerenti all'interpretazione dei risultati, possono insorgere altri tipi di problematiche relative ad aspetti più pratici.

Un possibile problema riguarda il cambiamento delle condizioni di lavoro normali di una base dati, o di un sistema in generale. Queste potrebbero mutare in modo considerevole in seguito, per esempio, a un aggiornamento di release, innescando così un periodo di transizione in cui il modello precedentemente elaborato non potrebbe più garantire le stesse prestazioni. In tal caso, sono necessari cicli di training perché la rete possa interpretare in modo corretto il nuovo comportamento del sistema.

Un altro aspetto riguarda le capacità predittive di un modello, da intendersi in senso generale. Più precisamente, ciò che si vuole predire è il degrado delle prestazioni. Potrebbe accadere, tuttavia, che le performance del sistema degradino in modo impulsivo, a seguito dell'esecuzione, per esempio, di un *job* particolarmente oneroso. Un caso del genere renderebbe del tutto vana l'analisi predittiva, in quanto eventi estemporanei, come può essere l'intervento umano, non sono prevedibili per definizione.

L'orizzonte temporale, ossia il numero di campioni futuri, costituisce un altro parametro critico. In linea di principio, prevedere un arco temporale di 1 giorno o 1 settimana, piuttosto che di 10 minuti, costituisce un vantaggio notevole. Nella pratica, con

l'aumentare dell'orizzonte temporale, l'errore di predizione tende a crescere, in molti casi.

Come anticipato nel paragrafo 2.3.1 a pagina 9, fattori molto importanti da tenere in considerazione sono i tempi di estrazione delle informazioni utili e di esecuzione del modello. Tali fattori influiscono in maniera considerevole sulla scalabilità della soluzione, soprattutto nella prospettiva futura di estendere la soluzione proposta anche ad altri sistemi.

Senza dubbio, il caricamento delle informazioni costituisce un *overhead*, in quanto i dati devono passare attraverso Elasticsearch per poter essere elaborati e, conseguentemente, visualizzati. Tuttavia, facendo riferimento in modo particolare alle reti implementate, il numero di informazioni richiesto e i tempi di esecuzione risultano essere ragionevoli [1], tali cioè da garantire l'acquisizione del risultato con tempi dell'ordine dei secondi. Ovviamente, volendo aumentare la complessità della rete e/o la quantità di informazioni da prelevare, opportune valutazioni, dal punto di vista dell'efficienza, risultano necessarie.

6.4 Sviluppi futuri

Come più volte evidenziato nel corso del capitolo 3 a pagina 11, la soluzione proposta è stata progettata con lo specifico intento di garantire la possibilità di estenderne il dominio e le funzionalità. In breve, si è cercato di realizzare una soluzione versatile e flessibile, caratteristiche che ben si accordano con la prospettiva di sviluppo.

Riguardo all'implementazione delle reti neurali, nel lavoro di tesi sono state considerate esclusivamente tre diverse implementazioni: due reti *feed-forward* e una rete *ricorrente*. La ricerca, nel campo delle reti neurali, è attualmente molto attiva, il che ha portato all'esplorazione e alla definizione di diverse topologie, oltre a quelle esaminate. Per il problema affrontato, potrebbero essere prese in considerazione reti neurali differenti, come, per esempio, le reti neurali convoluzionali (in inglese, *convolutional neural network*, CNN), che, nonostante siano progettate principalmente per il riconoscimento di immagini e video, si ritrovano anche in alcuni studi sulle serie temporali.

L'analisi svolta ha tenuto conto, inoltre, di un sottoinsieme delle metriche di performance rilevate dall'Oracle Enterprise Manager. Questi dati potrebbero essere integrati con altri tipi di informazioni inerenti al sistema analizzato (e.g., file di log), con il fine di descrivere in modo più accurato il suo stato e, auspicabilmente, effettuare delle predizioni più precise. Oppure, si potrebbe considerare un insieme di campioni più consistente, o metriche rilevate a una frequenza maggiore, aumentando in tal modo il livello di dettaglio.

[1] I tempi di addestramento della rete variano, invece, considerevolmente, dall'ordine di qualche minuto, per le reti più semplici, fino anche a un paio d'ore (facendo riferimento ai test eseguiti). Ovviamente bisogna tener conto anche degli parametri: numero di epoche, dimensione della rete eccetera.

Inoltre, si può pensare di combinare l'analisi di tipo predittivo con un'operazione di classificazione dei pattern individuati, in modo da poter discernere le situazioni problematiche da quelle normali, sfruttando le informazioni relative a eventi problematici, registrati solitamente all'interno di file di sistema.

Come si può facilmente intuire, gli aspetti da indagare sono innumerevoli e la possibilità di ospitare nuove metodologie di analisi costituisce, senza dubbio, uno degli aspetti più interessanti della soluzione proposta, insieme, chiaramente, alla capacità di predire lo stato evolutivo di un sistema.

Bibliografia

Bengio, Y., Simard, P. and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks* **5**(2): 157–166.
URL: *http://dx.doi.org/10.1109/72.279181*

Brownlee, J. (2019). A gentle introduction to autocorrelation and partial autocorrelation. 14 Aprile 2019.
URL: *https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation*

Burleson, D. (2016). Understanding oracle features and options. 10 Maggio 2019.
URL: *http://www.dba-oracle.com/art_so_oracle_standard_enterprise_edition.htm*

Croarkin, C. and Tobias, P. (2013). Nist/sematech e-handbook of statistical methods. 14 Aprile 2019.
URL: *https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm*

Hamilton, J. D. (1994). *Time Series Analysis*, Princeton University Press.

Haykin, S. S. (2009). *Neural Networks and Learning Machines*, Prentice Hall.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory, *Neural computation* **9**: 1735–80.

Hyndman, R. J. and Athanasopoulos, G. (2018). Forecasting: principles and practice, OTexts: Melbourne, Australia. 15 Aprile 2019.
URL: *https://otexts.com/fpp2*

Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy, *International Journal of Forecasting* **22**(4): 679–688.
URL: *http://www.sciencedirect.com/science/article/pii/S0169207006000239*

McCarthy, J. (2007). What is artificial intelligence? 4 Giugno 2019.
URL: *http://www-formal.stanford.edu/jmc/whatisai.pdf*

Mullins, C. S. (2010). Defining database performance. 7 Maggio 2019.
URL: *http://www.dbta.com/Columns/DBA-Corner/Defining-Database-Performance-70236.aspx*

Olah, C. (2015). Understanding lstm networks. 13 Aprile 2019.
URL: *http://colah.github.io/posts/2015-08-Understanding-LSTMs/*

Oracle (2015). Enterprise manager oracle database plug-in metric reference manual. Plug-in Release 12.1.0.7.

Scikit-learn developers (2019). Scikit-learn user guide, release 0.20.3. 12 Aprile 2019.
URL: *https://scikit-learn.org/0.20/_downloads/scikit-learn-docs.pdf*

Torch contributors (2018). Pytorch master documentation. 21 Aprile 2019.
URL: *https://pytorch.org/docs/stable/nn.html#lstm*

VMware (2019). What is it automation? 4 Giugno 2019.
URL: *https://www.vmware.com/topics/glossary/content/it-automation*

Weisstein, E. (2002). Box-and-whisker plot, MathWorld—A Wolfram Web Resource. 22 Aprile 2019.
URL: *http://mathworld.wolfram.com/Box-and-WhiskerPlot.html*

Wikipedia contributors (2018). Application service management — Wikipedia, the free encyclopedia. 12 Maggio 2019.
URL: *https://en.wikipedia.org/w/index.php?title=Application_service_management*