Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea Magistrale in Informatica
Curriculum in Informatica per il Management

MANUTENZIONE PREDITTIVA DI MACCHINARI INDUSTRIALI TRAMITE TECNICHE DI INTELLIGENZA ARTIFICIALE: UNA VALUTAZIONE SPERIMENTALE

Tesi di Laurea Magistrale in Sistemi Context-Aware

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
LORENZO BIAGIO LANZARONE

Correlatori:
Dott. GERARDO FABRIZIO
Dott. DAVIDE DI DONATO

Sessione III
Anno Accademico 2019/2020

Alla mia famiglia...

Indice

Abstract

Nella società è in corso un processo di evoluzione tecnologica, il quale sviluppa una connessione tra l'ambiente fisico e l'ambiente digitale, per scambiare dati e informazioni. Nella presente tesi si approfondisce, nel contesto dell'*Industria 4.0*, la tematica della *manutenzione predittiva* di macchinari industriali tramite tecniche di *intelligenza artificiale*, per prevedere in anticipo il verificarsi di un imminente guasto, identificandolo prima ancora che si possa verificare.

La presente tesi è divisa in due parti complementari, nella prima parte si approfondiscono gli aspetti teorici relativi al contesto e allo stato dell'arte, mentre nella seconda parte gli aspetti pratici e progettuali. In particolare, la prima parte è dedicata a fornire una panoramica sull'Industria 4.0 e su una sua applicazione, rappresentata dalla manutenzione predittiva. Successivamente vengono affrontate le tematiche inerenti l'intelligenza artificiale e la Data Science, tramite le quali è possibile applicare la manutenzione predittiva. Nella seconda parte invece, si propone un progetto pratico, ossia il lavoro da me svolto durante un tirocinio presso la software house *Open Data* di Funo di Argelato (Bologna). L'obiettivo del progetto è stato la realizzazione di un sistema informatico di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione, utilizzando tecniche di intelligenza artificiale. Il fine ultimo è l'integrazione di tale sistema all'interno del software *Opera MES* sviluppato dall'azienda.

Introduzione

Al giorno d'oggi l'intera società si trova di fronte ad un processo di evoluzione tecnologica che, attraverso l'utilizzo di macchine intelligenti collegate alla rete internet, sviluppa una costante connessione tra l'ambiente fisico e l'ambiente digitale, per scambiare dati e informazioni.

Alla base di questo studio vi è la volontà di approfondire, nel contesto dell'Industria 4.0, cioè la quarta rivoluzione industriale, la tematica della manutenzione predittiva di macchinari industriali tramite tecniche di intelligenza artificiale. Si tratta di una tipologia di manutenzione il cui scopo è quello di prevedere in anticipo il verificarsi di un imminente guasto, per permettere la rilevazione dell'anomalia ancora prima che si possa verificare, risparmiando in questo modo tempo e risorse.

La presente tesi è divisa in due parti distinte ma complementari, nella prima parte si approfondiscono gli aspetti teorici relativi al contesto e allo stato dell'arte, mentre nella seconda parte gli aspetti pratici e progettuali. In particolare, la prima parte è dedicata a fornire una panoramica sull'Industria 4.0 e su una sua applicazione, rappresentata dalla manutenzione predittiva. Successivamente vengono affrontate le tematiche inerenti l'intelligenza artificiale e la Data Science, tramite le quali è possibile applicare la manutenzione predittiva. Nella seconda parte invece, si propone un progetto pratico, ossia il lavoro da me svolto durante un tirocinio della durata di quattro mesi presso la software house *Open Data* di Funo di Argelato (Bologna), tra ottobre 2020 e febbraio 2021. L'obiettivo del progetto è stato la realizzazione di un sistema informatico di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezio-

ne, utilizzando tecniche di intelligenza artificiale. Il fine ultimo è l'integrazione di tale sistema all'interno di *Opera MES*, il software di Manufacturing Execution System sviluppato dall'azienda. I macchinari industriali oggetto dell'analisi si trovano presso l'azienda cliente *STS Tecnopolimeri* di Jesi (Ancona).

Nel progetto realizzato si sono seguite tutte le fasi definite dal modello di processo *CRISP-DM*, eseguendo innanzitutto un'analisi del dominio del problema, approfondendo il funzionamento delle presse a iniezione e del software MES. Successivamente è avvenuta la raccolta dei dati da utilizzare provenienti dai macchinari industriali, esportandoli dal database *Microsoft SQL Server* tramite apposite *query*. In particolare sono stati raccolti i dati relativi a sensori, difetti degli articoli prodotti, produzioni con guasto e produzioni corrette. Tali dai sono stati catalogati e analizzati con metriche statistiche e grafici utilizzando appositi script *Python* e librerie software dedicate. Successivamente si è eseguita un'analisi sulla correlazione dei dati dei sensori e sul clustering delle produzioni con guasto, con l'obiettivo di ridurre la quantità dei dati considerati selezionando i più significativi e assegnare una tipologia ai guasti non causalizzati per rilevare i *fermi macchina*. A causa della complessità del dominio del problema non è stato possibile identificare correttamente le tipologie di guasto e i fermi macchina tramite le tecniche di clustering utilizzate. Dopo aver selezionato i dati più rilevanti si sono costruiti cinque diversi *DataFrame* con logiche e complessità differenti, per essere utilizzati come input dei modelli di manutenzione predittiva. Successivamente si sono costruiti, allenati e testati tre modelli differenti utilizzando Support Vector Machine e due *1D Convolutional Neural Network* di classificazione e con autoencoder. In seguito, si sono valutati i risultati prodotti dai diversi DataFrame su tali modelli, selezionando la combinazione che presentava i risultati migliori, ossia il modello costruito con *Support Vector Machine*. Di tale modello si sono eseguiti test e valutazioni più approfondite, che hanno rilevato un'accuracy dell'84% e una predizione media del guasto con 2,5 ore di anticipo. Infine si è eseguito il rilascio del sistema, implementando il web server, la pagina di configurazione e definendo il modo in cui deve essere integrato in Opera MES, anche con la realizzazione di alcuni mockup. Si sono inoltre progettate e discusse le architetture del sistema per essere eseguito con gli approcci di cloud computing e di edge computing, evidenziandone vantaggi e svantaggi.

La tesi è strutturata in cinque capitoli, i primi due sono inclusi nella **prima parte** e gli ultimi tre nella **seconda parte**. Il contenuto di ogni capitolo è descritto di seguito:

- Nel **primo capitolo** si propone una panoramica sull'Industria 4.0 e sulla manutenzione predittiva. Innanzitutto, con alcuni cenni storici si percorre il passaggio dalla prima rivoluzione industriale ad oggi, per poi descrivere il significato e le caratteristiche proprie dell'Industria 4.0. Successivamente si evidenziano le tecnologie abilitanti, le azioni governative e i principali effetti e benefici portati dalla quarta rivoluzione industriale. Infine, nel contesto dell'Industria 4.0, si descrive lo scopo della manutenzione predittiva e i relativi utilizzi.

- Nel **secondo capitolo** si propone una panoramica sull'intelligenza artificiale e sulla Data Science, tramite le quali è possibile realizzare un modello di manutenzione predittiva. Dopo aver definito i concetti di *Machine Learning* e *Deep Learning*, si descrivono gli approcci di apprendimento supervisionato, non supervisionato e le reti neurali artificiali. Infine sono approfondite le tecniche di intelligenza artificiale utilizzate per la realizzazione del progetto descritto nella seconda parte dell'elaborato. In particolare gli algoritmi *Support Vector Machine*, *K-Means*, *Agglomerative Clustering* e *DBSCAN*, oltre alle *Convolutional Neural Network* e agli *Autoencoder*.

- Nel **terzo capitolo** si descrive la fase di progettazione del lavoro svolto durante il tirocinio, nel quale si è realizzato un sistema informatico di manutenzione predittiva di macchinari industriali per lo stampaggio plastico. Dopo aver definito gli obiettivi e le specifiche del progetto si presenta il modello di processo CRISP-DM, utilizzato durante l'intero svolgimento del progetto. Successivamente, vengono approfondite le prime quattro fasi di tale modello, cioè *Business Understanding*, *Data Understanding*, *Data Preparation* e *Modeling*. Per ogni fase sono descritti i task e le operazioni che si sono svolte, dal recupero e selezione dei dati, alla realizzazione del modello di manutenzione predittiva, passando per l'analisi e la manipolazione dei dati di interesse.

- Nel **quarto capitolo** si descrive la fase di implementazione del progetto, fornendo una panoramica delle tecnologie e degli strumenti utilizzati, descrivendone aspetti tecnici, caratteristiche principali e in che modo si sono applicati nel progetto.

Ad esempio, dal linguaggio di programmazione Python alle librerie software *Pandas*, *Scikit-learn* e *Keras*, passando per il framework *Flask* e la piattaforma di cloud computing *Heroku*. Infine sono riportati gli snippets più rilevanti del codice implementato, suddivisi in base al task e al file a cui appartengono.

- Nel **quinto capitolo** si descrive il processo di validazione del progetto realizzato, in particolare sono approfondite le ultime due fasi del modello CRISP-DM, cioè *Evaluation* e *Deployment*. Per entrambe sono descritti i task e le operazioni che si sono svolte, dalla valutazione dei risultati prodotti dai modelli di apprendimento alla selezione del migliore di essi, passando poi al rilascio del sistema. In quest'ultima fase si descrive la costruzione del web server e della pagina di configurazione, l'esecuzione del sistema sul cloud computing e la relativa architettura, oltre all'integrazione del progetto in Opera MES anche tramite mockup. Infine si propone una variante dell'architettura del sistema utilizzando un approccio di edge computing, analizzandone vantaggi e svantaggi rispetto al cloud computing. Un'ultima sezione è dedicata alla descrizione dei problemi riscontrati durante la realizzazione del progetto e alle relative soluzioni applicate.

Parte I

Contesto e stato dell'arte

"C'è vero progresso solo quando i vantaggi di una nuova tecnologia diventano per tutti."
Henry Ford

Capitolo 1

Industria 4.0 e manutenzione predittiva

Al giorno d'oggi l'intera società si trova di fronte ad un processo di evoluzione tecnologica che, attraverso l'utilizzo di macchine intelligenti collegate alla rete internet, sviluppa una costante connessione tra l'ambiente fisico e l'ambiente digitale. Vi è un crescente scambio di dati e informazioni, sulle quali possono essere eseguite analisi complesse, in grado di trovare soluzioni a problemi anche in tempo reale.

Ci troviamo nella cosiddetta quarta rivoluzione industriale, che ha preso il nome di *Industria 4.0* e che, secondo la *Commissione Europea*, è un'espressione che si riferisce alla trasformazione dell'intera sfera della produzione industriale avvenuta grazie alla fusione della tecnologia digitale e di internet con la manifattura tradizionale [1]. Il presente elaborato si colloca esattamente all'interno di questo tema, in particolare nell'ambito della manutenzione predittiva. Risulta quindi necessario mettere in evidenza alcuni aspetti:

- I principali eventi storici che hanno dato vita alla quarta rivoluzione industriale.
- Le tecnologie abilitanti dell'Industria 4.0, che integrano e innovano tecnologie esistenti per lo sviluppo di nuovi processi in chiave digitale.
- Le azioni governative, gli effetti, i benefici e i futuri scenari possibili dell'Industria 4.0. Affrontando inoltre i mutamenti dei processi produttivi e del rapporto uomo-macchina, anche in relazione alle nuove competenze professionali richieste.

- Una panoramica sulla manutenzione predittiva, un'applicazione che si colloca nell'ambito dell'Industria 4.0 e della quale è stato realizzato il progetto descritto nella seconda parte dell'elaborato.

1.1 Dalla prima alla quarta rivoluzione industriale

L'obiettivo di questo capitolo è quello di percorrere i principali eventi storici e i fenomeni che hanno portato alla nascita dell'Industria 4.0, a partire dalla prima rivoluzione industriale:

- **Prima rivoluzione industriale:** la prima rivoluzione industriale ha avuto inizio nel Regno Unito, in questo periodo si è sviluppato il passaggio dal lavoro manuale agricolo e artigianale ai primi processi produttivi. Si colloca tra la fine del Settecento e l'inizio dell'Ottocento, con l'avvento della macchina a vapore e del telaio meccanico. In questo periodo è cresciuta la qualità dei prodotti realizzati e sono diminuiti i grandi sforzi fisici richiesti ai lavoratori.

- **Seconda rivoluzione industriale:** durante la seconda metà dell'Ottocento ha inizio la seconda rivoluzione industriale, con l'avvento dell'elettricità e delle prime catene di produzione. Importanti aziende si sono sviluppate grazie alla tecnologia e all'inserimento degli operai nelle catene di montaggio, per svolgere lavori assolutamente ripetitivi. La produzione industriale di massa si è sviluppata anche a seguito del cosiddetto *Modello Fordista*. L'imprenditore americano *Henry Ford*, con la famosissima automobile *Model T*, divenne il principale emblema della produzione in serie. Sono nate nuove comunità vicine alle città industriali, composte da famiglie con un solo percettore di reddito, da cui è emersa la necessità di sviluppare un nuovo sistema di *welfare*. Con la prima e la seconda rivoluzione industriale si sono prodotti quindi grossi cambiamenti sociali determinati dall'aumento dei redditi medi e dalle migliori condizioni lavorative e di vita.

- **Terza rivoluzione industriale:** la terza rivoluzione industriale si colloca nella seconda metà del Novecento. Sviluppatasi grazie alla digitalizzazione, all'elettronica

e all'automazione, l'industria si è dotata di macchinari in grado di svolgere operazioni sempre più complesse e in autonomia. Circuiti elettronici programmabili hanno sostituito relè e interruttori. Grazie a macchine a controllo numerico, robot e sistemi di trasporto è diventato possibile modificare le prestazioni dei macchinari e gestire la complessità dei dati e dei processi, rendendo maggiormente flessibili le linee di produzione. Comincia l'analisi dei dati digitali con i quali migliorare la gestione dei sistemi produttivi e la loro progettazione. Inoltre avviene un continuo miglioramento dei processi, dei prodotti e delle condizioni lavorative.

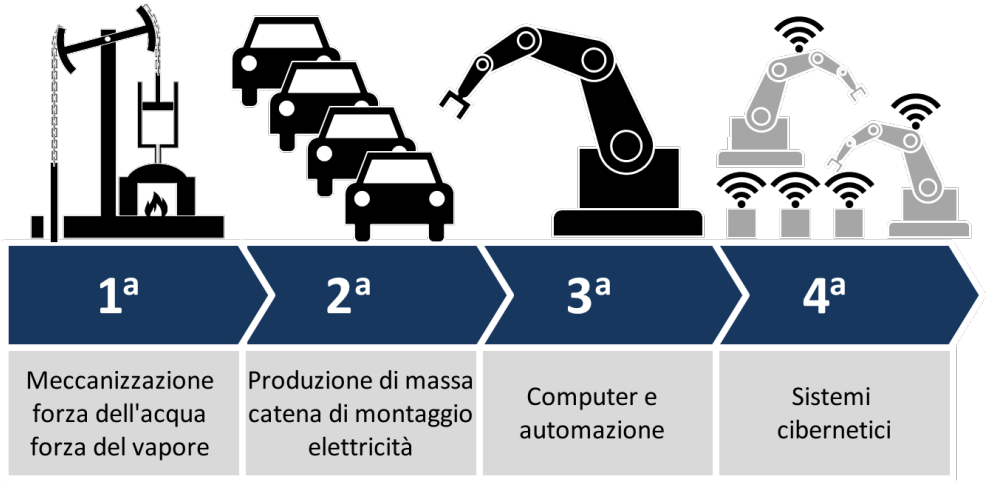

Figura 1.1: Le quattro rivoluzioni industriali

Il susseguirsi delle tre rivoluzioni appena descritte ha consentito il raggiungimento della situazione attuale. Il progresso è però in continua evoluzione e diversi impulsi di natura tecnologica, economica, politica, sociale, ambientale e legislativa hanno portato l'industria ad avanzare ulteriormente attraverso un nuovo paradigma. In particolare, alcuni dei fattori che hanno portato alla **quarta rivoluzione industriale** sono descritti di seguito:

- L'aumento della necessità di customizzazione dei prodotti, al fine di soddisfare le preferenze dei singoli consumatori anziché della massa.

- La progressiva riduzione del *time to market*, cioè il tempo che intercorre dall'ideazione di un prodotto alla sua effettiva commercializzazione. Questo ha comportato la necessità di una più rapida capacità di innovare al fine di progettare, produrre ed immettere sul mercato beni e servizi in tempi brevi.

- Aumento della versatilità e della velocità dei processi produttivi, in modo tale da produrre prodotti a costi più contenuti ma con una maggiore qualità.

- Infine, le innovazioni tecnologiche come ad esempio internet, applicazioni, software MES, intelligenza artificiale, Machine Learning, social network, dispositivi mobili e stampanti 3D. Queste e molte altre tecnologie hanno portato a nuove e innovative opportunità per la crescita di automazione, digitalizzazione, networking, miniaturizzazione, produzione e fornitura [2].

1.2 Panoramica sull'Industria 4.0

Il termine *Industria 4.0* è apparso per la prima volta nel 2011 in Germania, in occasione della *Fiera Annuale di Hannover*, durante la quale *Henning Kagermann*, *Wolf-Dieter Lukas* e *Wolfgang Wahlster* proposero un progetto di investimenti su infrastrutture, enti di ricerca, sistemi energetici, istruzione ed aziende [3]. L'obiettivo era quello di riportare il settore manifatturiero tedesco ai vertici mondiali, ammodernandone il tessuto produttivo e rendendo la Germania nuovamente competitiva a livello mondiale. Si tratta quindi di un termine piuttosto recente, coniato per descrivere la quarta rivoluzione industriale. È possibile affermare che l'Industria 4.0 è caratterizzata dalla digitalizzazione avanzata, dall'integrazione tra i processi produttivi e logistici e dall'utilizzo di dispositivi intelligenti. Per lo sviluppo delle azioni necessarie al compiersi dell'Industria 4.0, è compito innanzitutto dei governi stimolare un movimento culturale, economico e sociale, capace di trasformare la scena di ogni dimensione che ci circonda, stimolando le imprese, formando le competenze richieste e investendo sull'innovazione digitale. Le azioni governative sono necessariamente affiancate dalle *big corporation* della tecnologia e delle merci, che si occupano di attuare e modificare continuativamente il processo che sta riscrivendo le relazioni tra esseri umani e dispositivi, oltre che tra dispositivi stessi. Alcune tecnologie

Figura 1.2: Rappresentazione grafica dell'Industria 4.0

di supporto, chiamate *tecnologie abilitanti*, hanno raggiunto costi estremamente bassi e larga diffusione, inoltre si è ridimensionato il livello di competenza necessario per maneggiarle. Oggi l'informazione è al centro della nostra vita, ogni componente produttivo e ogni bene prodotto possono essere collegati in tempo reale tramite internet ad un modello in cloud che ne segue il processo, raccoglie i dati e li rielabora simultaneamente, per governare sistemi produttivi remoti e riprocessare le catene del valore di prodotti e servizi. Alla base dell'Industria 4.0 sono presenti quattro *enabler* tecnologici tra loro interdipendenti, descritti di seguito:

- **Cyber-Physical Production System:** si tratta di un sistema informatico che è in grado di interagire in modo continuo con il sistema fisico in cui opera. Questo significa che i computer e le reti sono in grado di monitorare e gestire i processi fisici di fabbricazione.

- **Internet of Things:** è la capacità che hanno gli oggetti e le macchine a comunicare tra loro e con l'essere umano per risolvere un problema. L'integrazione di questa tecnologia permette agli oggetti di cooperare, anche in maniera indipendente rispetto alla componente umana.

- **Internet of Service:** indica la convergenza di altri due concetti, *Web 2.0* e *Service-Oriented Architecture (SOA)*. La loro intersezione mira al riutilizzo e al-

l'integrazione di risorse e servizi esistenti tramite la rete internet, per semplificare interoperabilità e azioni [4].

- **Smart Factory:** è il risultato dell'utilizzo delle tre precedenti tecnologie che insieme costituiscono un sistema capace di far dialogare il mondo fisico con uno virtuale e che assiste le persone e le macchine nell'esecuzione dei loro compiti.

Ricercatori e operatori dell'Industria 4.0 hanno individuato i principali vantaggi raggiunti dalla quarta rivoluzione industriale. Tali vantaggi possono essere riassunti nei tre seguenti obiettivi:

- **Ottimizzazione:** il primo vantaggio è quello dell'ottimizzazione della produzione, riducendo al minimo i tempi morti che da sempre rappresentano la limitazione più forte ai processi di innovazione. Aumenta inoltre la produttività e la qualità dei prodotti, mentre si riducono gli scarti e i guasti ai macchinari.

- **Customizzazione:** il secondo vantaggio è la cosiddetta customizzazione, cioè il favorire una produzione flessibile e orientata ai bisogni del consumatore, preservando contemporaneamente l'efficienza della produzione di massa.

- **Innovazione:** il terzo vantaggio riguarda lo stimolo ad attuare nuova ricerca, quindi ulteriore innovazione, per lo sviluppo di nuove competenze, metodi e approcci operativi.

Accanto a questi vantaggi emergono comunque delle criticità dovute all'impatto del cambiamento che dall'industria si espande al sistema economico, politico e all'intera società, primo tra tutti il mutamento del mercato del lavoro. Oggi ci troviamo all'interno di questo contesto ed occorre ripensare il sapere, il saper fare e il saper essere [5].

1.2.1 Tecnologie abilitanti

L'Industria 4.0 è resa quindi possibile dall'utilizzo in ambito produttivo delle componenti descritte precedentemente e da diverse tecnologie abilitanti, con lo scopo di ammodernare il tessuto produttivo, adeguandolo alle nuove opportunità [6]. L'implementazione e l'integrazione di diverse tecnologie e strategie mira a rendere il processo

produttivo più snello ed efficiente, facendo leva sull'innovazione tecnologica che permette di creare legami e relazione fra i vari oggetti ed impianti presenti nello stabilimento produttivo, costruendo la cosiddetta *fabbrica digitale* [7]. Quello delle tecnologie abilitanti è un ambito in continua evoluzione non solo per lo sviluppo delle singole tecnologie ma soprattutto per la nascita di nuove interazioni tra tecnologie diverse che danno origine a nuove modalità di utilizzo delle stesse [8]. Allo scopo quindi di *fermare l'immagine*, nei prossimi capitoli si descrivono le nove macro categorie indicate dal *Governo Italiano*, più nello specifico dal *Ministero dello Sviluppo Economico (MISE)*, nella presentazione del *Piano nazionale Industria 4.0*, introdotto nel 2017 [9]. Tali tecnologie sono riassunte nella figura 1.3 e sono le medesime individuate da uno studio di *Boston Consulting Group*, multinazionale statunitense di consulenza strategica [10].

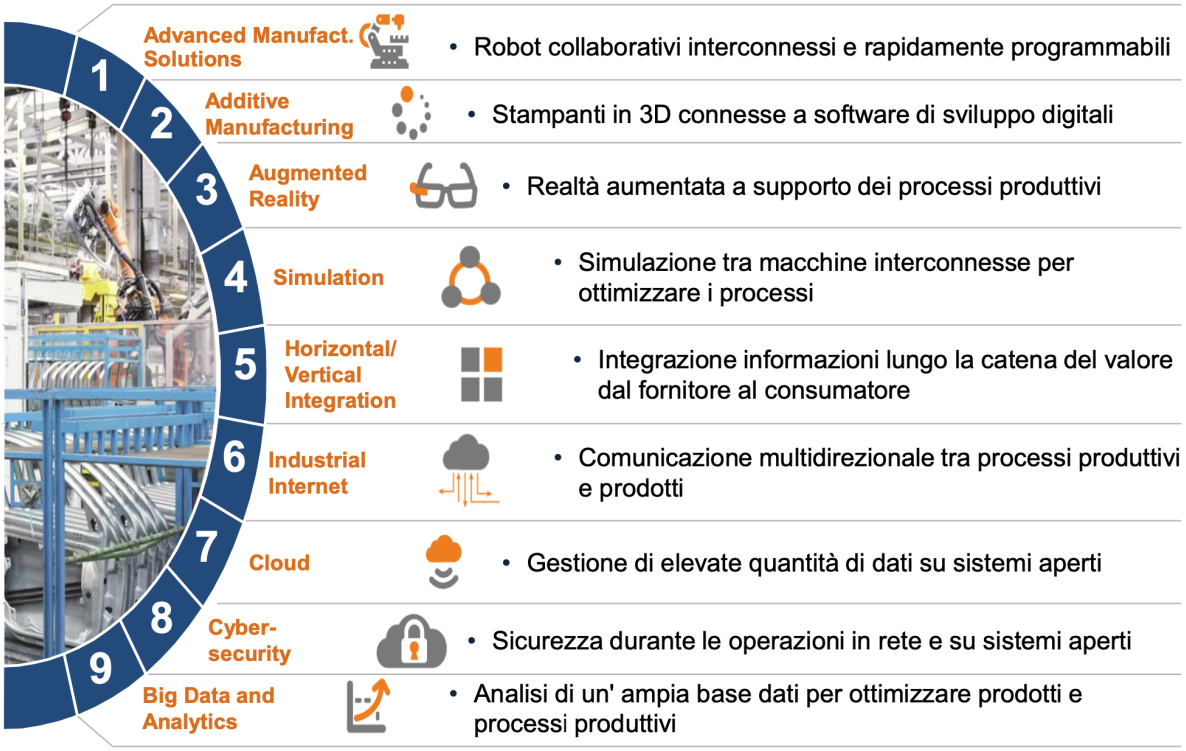

Figura 1.3: Tecnologie abilitanti definite dal Piano nazionale Industria 4.0

1.2.1.1 Robotica collaborativa

La robotica collaborativa, nota anche con il termine inglese *advanced manufacturing solutions*, consiste nell'utilizzo di robot collaborativi, detti *cobot*, che grazie a sensori sempre più potenti e intelligenti e software automatizzati, sono in grado di gestire i propri movimenti. Con la robotica collaborativa si sviluppa il rapporto uomo-macchina, creando una collaborazione reale fra il cobot e l'essere umano nello stesso spazio lavorativo, consentendo al cobot di apprendere da quest'ultimo. Il cobot accompagna l'operatore nelle attività lavorative più complesse, svolgendo compiti pericolosi e pesanti senza fatica e con accuratezza. Da parte sua l'operatore ha capacità cognitive che nella collaborazione con il cobot impiega laddove l'organizzazione dei processi condivisi prevede un certo grado di interpretazione del contesto. Le nuove tecnologie consentono di eseguire un monitoraggio in tempo reale sia dell'uomo sia dei cobot attraverso delle reti di sensori installate nelle aree di lavoro condiviso oppure tramite dispositivi abilitati alla fruizione della realtà aumentata. Altre tecnologie che saranno descritte nei prossimi capitoli, come l'Internet of Things e l'intelligenza artificiale, supportano lo sviluppo della robotica collaborativa.

1.2.1.2 Manifattura additiva

La manifattura additiva, nota anche con il termine inglese *additive manufacturing*, è un processo usato per realizzare un oggetto attraverso l'uso di stampanti 3D connesse a software di sviluppo digitali. Questa tecnologia crea oggetti tridimensionali legando uno strato alla volta di materiale fuso o parzialmente fuso come polvere di metallo, materiali termoplastici, ceramiche, compositi, vetro e addirittura materiali commestibili come il cioccolato. Un software CAD (acronimo di *Computer-Aided Design*) definisce digitalmente l'oggetto. Il file del progetto, memorizzato in appositi formati, contiene le informazioni per guidare la costruzione di tale oggetto da parte della stampante 3D, in grado di unire strati sottilissimi di materiale con una elevata precisione [11].

1.2.1.3 Realtà aumentata

La realtà aumentata, nota anche con il termine inglese *augmented reality*, offre la possibilità di aggiungere alla realtà ulteriori informazioni e dimensioni potenziando la quantità di dati e di opportunità visibili dall'occhio umano. Mentre la realtà virtuale ha l'obiettivo di rimpiazzare completamente il mondo reale con quello virtuale, la realtà aumentata mira ad aggiungere elementi virtuali al mondo reale. La tecnologia offerta dalla realtà aumentata quindi combina oggetti reali e virtuali in un ambiente reale, operando in modo interattivo, in tre dimensioni ed in *real time*. I campi di applicazione di tale tecnologia sono infiniti: dall'automotive al settore manifatturiero, passando per sistemi di packaging intelligenti. Nel campo della logistica, la realtà aumentata è utile per localizzare velocemente i prodotti in magazzino, oppure per verificare in tempo reale la conformità degli ordini. Nel settore industriale i visori ottici aiutano gli operatori a individuare le componenti guaste o difettose di un macchinario, mentre nel marketing la realtà virtuale consente di testare in anteprima aspetti estetici e funzionali dei prodotti, collocandoli virtualmente nell'ambiente circostante. Sono molti gli strumenti che permettono di usufruire di questa tecnologia, come visori ottici, occhiali intelligenti, manuali interattivi ma anche smartphone e tablet [12].

1.2.1.4 Simulazione

La simulazione attiene al concetto di *digital twin*, cioè gemello digitale. Grazie alle tecnologie abilitanti dell'Industria 4.0, in particolare IoT, Cloud Computing e Big Data Analytics, è possibile generare la versione virtuale di un processo, prodotto o servizio. Il sistema informatico, nel caso del processo produttivo, può quindi simulare diversi scenari relativi alla linea di produzione, in modo da scegliere quello che ottimizza le prestazioni. È inoltre possibile simulare il comportamento di sistemi complessi, riuscendo a prevedere ad esempio guasti meccanici, riducendo così inefficienze e costi. Gli operatori possono testare ed ottimizzare le impostazioni dei macchinari all'interno di un ambiente virtuale prima che il prodotto entri effettivamente in produzione, permettendo di incrementarne notevolmente la qualità. L'utilizzo si presta quindi a diversi scopi, fra i quali la manutenzione predittiva.

1.2.1.5 Integrazioni digitali

Le tecnologie relative all'Industria 4.0, hanno permesso l'integrazione delle informazioni nel processo produttivo dal fornitore al consumatore. Questo è possibile sviluppando la connessione fra i diversi processi produttivi, la cosiddetta *integrazione orizzontale*, oltre che tramite l'*integrazione verticale*, cioè la connessione della produzione con le altre aree aziendali, dalla logistica interna ai servizi post-vendita. L'integrazione orizzontale sfrutta soluzioni intelligenti all'interno dell'impianto produttivo, tramite il Cyber Physical Production System si raggiungono elevati livelli di performance nei processi di produzione. Solitamente l'integrazione orizzontale si sviluppa su diversi livelli:

- **Nello stesso impianto di produzione:** una linea di produzione interconnessa sarà in grado di rispondere dinamicamente allo stato di ogni macchina per aumentare l'efficienza, riducendo i tempi di inattività attraverso la manutenzione predittiva.

- **Tra diversi impianti di produzione:** nel caso in cui un'azienda disponga di più stabilimenti di produzione distribuiti, l'integrazione orizzontale tra i sistemi di gestione della produzione, i cosiddetti *Manufacturing Execution System*, è in grado di smistare le attività produttive tra i diversi impianti in base alle necessità.

- **Attraverso la catena di distribuzione:** si attua una collaborazione automatizzata e uno scambio di informazioni tra soggetti esterni all'azienda, dai fornitori che alimentano il processo di produzione ai distributori che immettono sul mercato i prodotti finiti.

Nell'Industria 4.0, l'integrazione verticale consente di collegare tutti i livelli all'interno della fabbrica digitale, in maniera trasversale rispetto alla sua struttura gerarchica. Dalla logistica interna alla produzione, fino alla vendita e al marketing, tutti i livelli si scambiano continuamente dati e informazioni utili. Grazie a questa interconnessione, proprio tali dati diventano di supporto per le decisioni strategiche e tecniche da compiere, apportando un vantaggio competitivo all'azienda [13].

Figura 1.4: Esempio di utilizzo delle tecnologie nell'Industria 4.0

1.2.1.6 Industrial Internet of Things

Con il termine *Internet of Things*, in italiano *internet delle cose*, ci si riferisce alla rete di apparecchiature e dispositivi connessi ad internet, cioè tutti gli elementi che possono essere dotati di intelligenza e autonomia grazie a sensori, microprocessori e software. Potenzialmente ogni oggetto fisico può generare e condividere dati sul proprio stato e quello dell'ambiente fisico circostante, diventando così *smart*. Inoltre, grazie alle reti wireless è possibile raccogliere e condividere tali dati con enorme velocità. Attraverso un indirizzo IP che ne permette l'identificazione univoca sulla rete e la capacità di scambiare dati attraverso essa è potenzialmente possibile collegare ad internet qualunque tipologia di oggetto, si stima che entro il 2025 saranno oltre 50 miliardi i dispositivi connessi a internet nel mondo. In questo modo è possibile disporre di informazioni un tempo inimmaginabili e sfruttando processi automatizzati si semplifica la vita degli individui. Ad esempio i dispositivi per la domotica domestica, la *smart road* in grado di comunicare con le automobili, oltre al crescente invio di informazioni attraverso dispositivi mobili dotati di sensori come smartphone e smartwatch. Oltre ad Internet of Things si utilizza il

termine *Industrial Internet of Things*, per distinguere le tecnologie utilizzate nel mondo consumer e quelle dedicate al mondo industriale, nel quale è necessario un maggiore livello di integrazione nei processi produttivi. I campi di applicazione di tale tecnologia sono molteplici, dai processi produttivi alla logistica e all'efficienza energetica. Specialmente in ambito industriale viene fatto largo uso di sensori, in grado di registrare dati di ogni genere per supportare l'efficienza della produzione. Ad esempio come nel caso del progetto di manutenzione predittiva proposto nella seconda parte dell'elaborato, nel quale vengono manipolati i dati registrati dai sensori posizionati in zone diverse di una pressa a iniezione. I punti maggiormente critici dell'Internet of Things riguardano la sicurezza e la privacy dei dati critici, ambiti nei quali le aziende devono prestare continua attenzione.

1.2.1.7 Cloud Computing

Con il termine *Cloud Computing*, si intende la fornitura di diversi servizi di calcolo tramite la rete internet, cioè nel cosiddetto *cloud* [14]. Tali risorse messe a disposizione da provider specializzati includono strumenti come ad esempio archiviazione e analisi dei dati, server, database e software applicativo. Le informazioni e le risorse a cui si accede sono localizzate nel cloud, uno spazio virtuale implementato in apposite server farm, anziché mantenere l'archiviazione e la computazione su un dispositivo locale. In questo modo è possibile la gestione e l'elaborazione di elevate quantità di dati, poiché le caratteristiche tecniche delle macchine offerte dai servizi di cloud computing sono particolarmente performanti. Grazie a questa tecnologia, un'azienda può utilizzare anche solo temporaneamente le risorse offerte dal cloud, in base alle proprie esigenze. Si parla di soluzioni *SaaS* (acronimo di *Software as a Service*), non è più necessario effettuare ingenti investimenti in infrastrutture informatiche, rinnovamento costante dell'hardware e manutenzione del sistema, tutte attività di cui si occupa il provider che gestisce il servizio. I vantaggi offerti dal cloud sono notevoli, accesso più rapido ad un'infrastruttura informatica esistente, supporto ai processi aziendali, elasticità e scalabilità in base alle esigenze, riduzione dei costi legati all'ICT e risparmio energetico. Il principale svantaggio riguarda invece la privacy e la sicurezza, poiché i dati non vengono più archiviati ed elaborati localmente, di conseguenza l'azienda non ne ha più il completo controllo. Nel progetto

di manutenzione predittiva realizzato e descritto nella seconda parte dell'elaborato si è utilizzata la piattaforma di cloud computing Heroku [15].

1.2.1.8 Sicurezza informatica

Tutte le tecnologie abilitanti non possono prescindere da interventi di sicurezza informatica, spesso indicata con il termine inglese *cybersecurity*. Si tratta dell'insieme dei mezzi e delle tecnologie per la protezione dei sistemi informatici in termini di disponibilità, confidenzialità e integrità, oltre all'autenticità delle informazioni. Come descritto precedentemente, poiché l'Industria 4.0 si basa sulla costante interconnessione delle componenti interne all'azienda con i propri fornitori e clienti tramite la rete internet, si crea un elevato scambio di dati. La tematica della sicurezza informatica diventa quindi imprescindibile, poiché se da un lato è possibile usufruire degli innumerevoli vantaggi degli interscambi di informazioni in rete, dall'altro si è esposti a furti di dati e sabotaggi. È quindi necessaria la gestione della sicurezza informatica, definendo le pratiche utilizzate in azienda per garantire la difesa della riservatezza, disponibilità e integrità delle risorse informatiche da possibili vulnerabilità e minacce. A livello internazionale, la sicurezza informatica aziendale è regolata da apposite normative come quelle proposte dall'ISO, in modo da standardizzare le modalità di protezione dei sistemi informatici. Si tratta indubbiamente di una tematica vasta e in continua evoluzione. Con lo sviluppo delle tecnologie deve procedere parallelamente anche lo sviluppo della cybersecurity per prevenire e affrontare eventuali danni al sistema informatico.

1.2.1.9 Big Data Analytics

L'enorme mole di dati prodotti dalla fabbrica digitale è caratterizzata da complessità, variabilità e volume. È necessario che tali dati vengano gestiti in modo adeguato. I dati sono strettamente correlati a fatti, sono potenzialmente riproducibili e possono fare parte di strutture informative più vaste come ad esempio *database* e *data warehouse*. In informatica, un dato è un valore, rappresentato in bit, che può essere elaborato da un calcolatore elettronico e archiviato in modo permanente su una memoria. Con il termine *Big Data* si indica un volume enorme di dati, strutturati e non strutturati, impossibili da analizzare senza l'utilizzo di software e calcolatori elettronici. Si tratta di dati che

possono provenire da fonti diverse, ad esempio social network, transazioni economiche o, come accennato precedentemente, dalla fabbrica digitale. Con Big Data Analytics si intendono tutti i processi e le tecnologie finalizzati all'acquisizione, memorizzazione, integrazione, preparazione e pulizia dei Big Data. Il principale scopo della Big Data Analytics riguarda ovviamente l'analisi di grandi dataset per ricavare da essi correlazioni e informazioni utili nel minor tempo possibile. Tali operazioni possono essere svolte tramite apposite piattaforme o librerie software. Si tratta di processi di modellazione, analisi ed interpretazione. Una volta avvenuto questo procedimento l'azienda è in possesso di vere e proprie informazioni attraverso le quali è possibile realizzare applicazioni complesse come ad esempio la manutenzione predittiva.

1.2.2 Azioni governative

Come descritto precedentemente, è compito dei Governi nazionali incentivare e programmare l'implementazione dell'Industria 4.0, investendo risorse pubbliche per la realizzazione di obiettivi anche a lungo termine. Prima di approfondire il caso italiano, si descrivono di seguito alcune delle principali iniziative governative attuate all'estero [16]:

- **Germania:** l'iniziativa del Governo tedesco, denominata *Zukunftsprojekt Industrie 4.0*, inizia nel 2011 nell'ambito del più ampio piano d'azione *High-Tech Strategy 2020*. L'obiettivo era la definizione di una strategia di digitalizzazione del settore manifatturiero tramite progetti di innovazione durante il successivo decennio. Gli elementi principali del piano erano: tecnologie, sinergie, governance e finanziamenti. Si è istituita una stretta cooperazione tra politica, industria, centri di ricerca e sindacati, in modo da aumentare la consapevolezza sulle soluzioni offerte dall'Industria 4.0.

- **Francia:** il Governo francese lanciò nel 2015 il piano *Industrie du futur*, che delineava i tratti della futura politica industriale del Paese. Esso era basato sulla cooperazione tra l'industria e la scienza, definendo cinque componenti principali: tecnologie all'avanguardia, supporto alle aziende nell'adozione delle nuove tecnologie, elevata formazione del personale, cooperazione internazionale per gli standard industriali e promozione dell'industria francese del futuro.

- **Regno Unito:** per definire una politica con lo scopo di sostenere la crescita e la resilienza della produzione inglese, nel 2013 il Governo britannico ha presentato l'iniziativa *Future of Manufacturing*. Tale iniziativa includeva una visione a lungo termine del settore manifatturiero del Regno Unito fino al 2050.

- **Unione Europea:** nel 2014 la Commissione Europea ha lanciato una partnership tra settore pubblico e settore privato chiamata *Factories of the Future*. Tale iniziativa rientrava nel più ampio programma *Horizon 2020*, che disponeva di un budget di finanziamento di quasi 80 miliardi di euro in sette anni.

- **Stati Uniti d'America:** per preparare gli Stati Uniti alla quarta rivoluzione industriale, nel 2011 il Governo ha avviato una serie di discussioni, azioni e raccomandazioni a livello nazionale denominate *Advanced Manufacturing Partnership (AMP)*. Il piano strategico includeva imprese industriali e big corporation del settore ICT, oltre a centri di ricerca e università, con l'obiettivo di innovare la manifattura e aumentarne la capacità occupazionale.

- **Giappone:** per realizzare la sua strategia *Super Smart Society*, il Governo giapponese avviò nel 2015 il *5th Science and Technology Basic Plan* a sostegno del settore manifatturiero del Paese.

- **Cina:** il Governo cinese introdusse nel 2015 il piano *Internet Plus*, oltre alla strategia *Made in China 2025*, il cui obiettivo era il rafforzamento dell'industria cinese. Quest'ultimo è simile al progetto Industrie 4.0 tedesco, definendo un piano di lungo periodo.

Si descrivono di seguito le iniziative governative attuate in **Italia**. La *Commissione attività produttive, commercio e turismo* della Camera dei Deputati avviò nel febbraio del 2016 una indagine conoscitiva con l'intento di sviluppare una strategia italiana per l'Industria 4.0, attraverso una migliore definizione del quadro normativo necessario a promuoverne la realizzazione. Successivamente, è stato predisposto il *Piano nazionale Industria 4.0*, presentato pubblicamente a Milano il 21 settembre 2016 dal Ministro dello Sviluppo Economico Carlo Calenda e dal Presidente del Consiglio Matteo Renzi. Le linee guida del Governo Italiano, contenute nel piano valido per il periodo 2017-2020, si

articolano in investimenti pubblici e privati per le tecnologie relative all'Industria 4.0, sviluppo di specifiche competenze professionali e della ricerca, realizzazione delle infrastrutture abilitanti, promozione di strumenti di supporto alle imprese e sensibilizzazione sull'importanza dell'Industria 4.0 nel settore pubblico e nel settore privato. A partire dal 2017, Italia, Germania e Francia, attraverso i relativi piani strategici nazionali, hanno avviato una cooperazione trilaterale per promuovere la digitalizzazione del settore manifatturiero e per sostenere gli sforzi dell'Unione Europea in quest'ambito. Poiché l'Italia si presenta strutturalmente diversificata e caratterizzata da un tessuto industriale ricco di piccole e medie imprese (PMI), attraverso il Piano nazionale Industria 4.0 ha dovuto supportare questo tipo di industrie tramite appositi sgravi fiscali, per un positivo avviamento al digitale. Per dare continuità al Piano nazionale Industria 4.0 è stato definito il *Piano nazionale Transizione 4.0*, che ha ricevuto in eredità le modalità tecniche ed operative del precedente e sarà valido fino al 2023. Si tratta di un importante investimento di circa 24 miliardi di euro, per una misura che diventa strutturale e su cui si fonda il *Recovery Fund* italiano. I due obiettivi principali del novo piano sono: stimolare gli investimenti privati e dare stabilità e certezze alle imprese [17].

1.2.3 Effetti e benefici

L'Industria 4.0 ha avviato un cambiamento che ha prodotto effetti in molti settori, aumentando la loro digitalizzazione attraverso tecnologie avanzate ed in continua evoluzione. L'utilizzo di robot, di realtà aumentata e delle altre tecnologie descritte precedentemente hanno ridefinito le fasi del processo produttivo, dalla progettazione alla distribuzione, rendendo più semplici e sicuri tutti i passaggi e garantendo minori rischi durante la lavorazione, maggiore qualità e minore costo del prodotto finale. Inoltre le aziende possono utilizzare specifici algoritmi di analisi dei dati e di intelligenza artificiale a supporto dell'attività decisionale. Anche il contesto sociale è stato favorito dall'Industria 4.0, gli individui ne possono beneficiare in ambito lavorativo e nelle attività quotidiane grazie a strumenti moderni e funzionali. Ad esempio attraverso i dispositivi wearable o tramite sistemi embedded e software dedicati è possibile connettere qualsiasi oggetto, dall'automobile agli elettrodomestici per realizzare la cosiddetta *smart home* e in prospettiva più ampia la *smart city*. Si tratta di un'intera comunità gestita digitalmente per

assicurare efficienza ed elevata qualità della vita, in grado di comunicare direttamente con la fabbrica digitale. Come si evince, i benefici sono indubbiamente molteplici, in un contesto che andrà sempre più a evolversi. Ma come nella maggior parte dei casi, gli effetti prodotti da un cambiamento di tale portata richiedono dei costi.

Come accennato nei capitoli precedenti, uno degli effetti più critici riguarda il cambiamento del mondo del lavoro. Lo studio *The Future of Jobs Report 2020* realizzato dal *World Economic Forum* affronta questa tematica attraverso l'intervista di 300 importanti aziende. L'84% dei manager consultati è favorevole alla digitalizzazione del lavoro, con una grande espansione dello smart working che potrebbe coinvolgere fino al 44% dei lavoratori. Secondo le aziende consultate nell'indagine, è necessario creare un senso di comunità, connessione e appartenenza attraverso gli strumenti digitali. Si prevede che il tasso di automazione operato dalle macchine passerà dal 33% del 2020 al 47% nel 2025, a fronte di una riduzione dell'attività umana che passerà dal 67% del 2020 al 53% nel 2025.

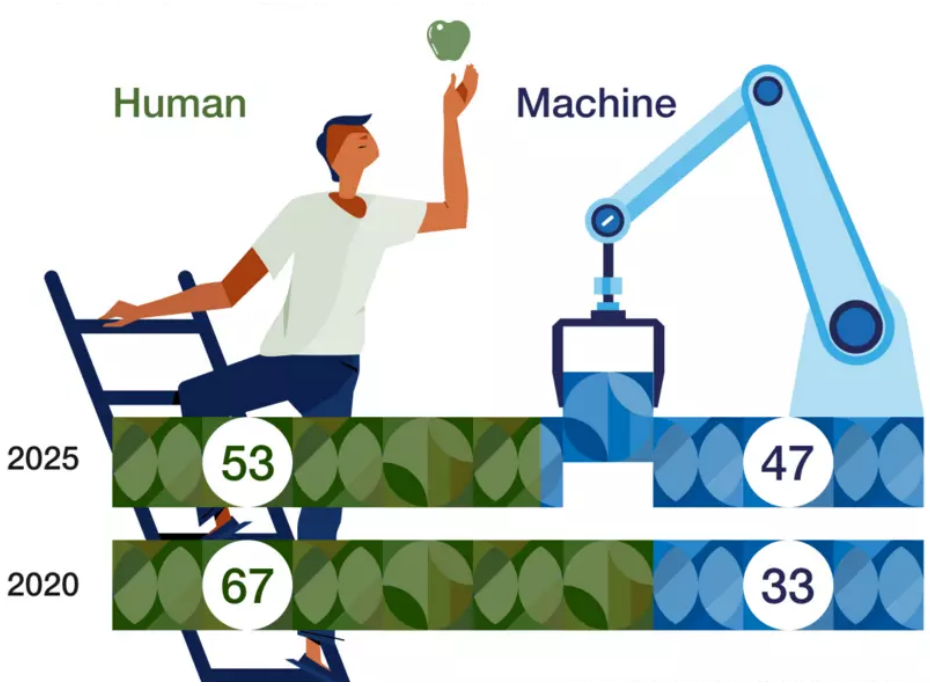

Figura 1.5: Tasso di automazione tra il 2020 e il 2025

L'adozione dei processi di digitalizzazione e automazione in atto hanno subito un'accelerata per fronteggiare la situazione critica dovuta alla pandemia di COVID-19 e le sue conseguenze, ma le ricadute della recessione economica hanno creato problemi nei mercati e nel mondo del lavoro. Si prevede che entro il 2025 i ruoli più routinari caleranno dal 15,4% della forza lavoro al 9% e che le professioni emergenti cresceranno dal 7,8% al 13,5%. 85 milioni di posti di lavoro potrebbero essere sostituiti a seguito di una diversa suddivisione del lavoro tra esseri umani e macchine, mentre sarebbero 97 milioni i posti di lavoro relativi alle nuove professioni necessari ai nuovi assetti organizzativi aziendali. Il 43% delle aziende prevede di ridurre la propria forza lavoro a causa dell'integrazione di nuove tecnologie nei processi di produzione, il 41% di espandere il proprio utilizzo di servizi esterni, mentre il 34% dovrà inserire un maggior numero di lavoratori specializzati per gestire e utilizzare le nuove tecnologie adottate.

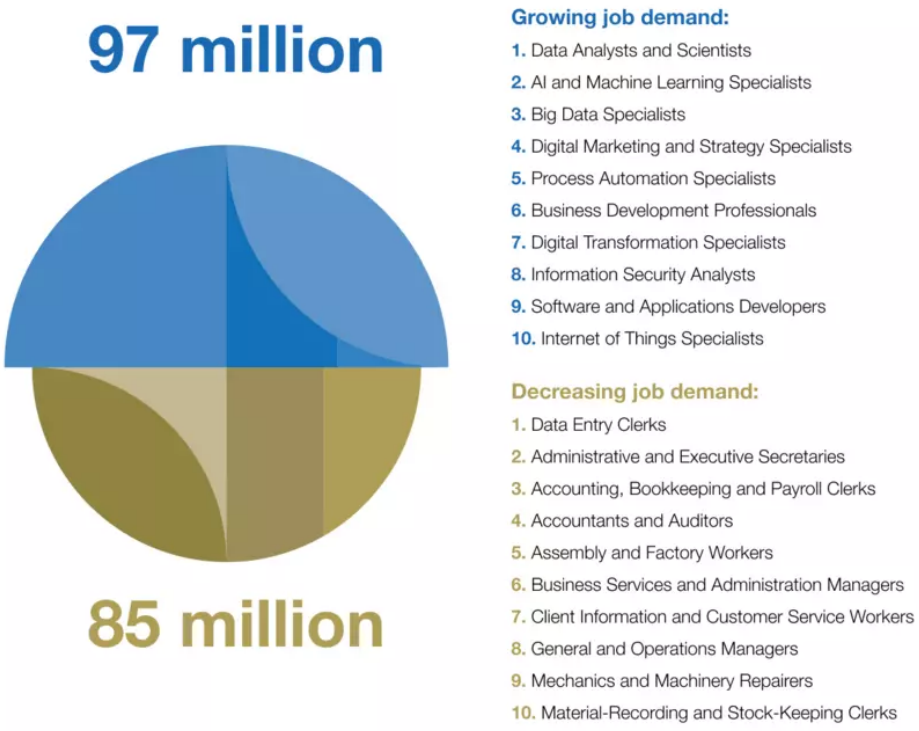

Figura 1.6: Professioni più e meno ricercate entro il 2025

Si comprende quindi che la domanda di professioni emergenti, specialmente del settore ICT ad alto livello di digitalizzazione, è destinata a crescere. Con l'adozione di nuove tecnologie nei processi di produzione di beni e servizi cresce l'esigenza di professioni in grado di maneggiare le tecnologie descritte dei capitoli precedenti come Internet of Things, robotizzazione, Big Data Analytics, intelligenza artificiale, Cloud Computing, sicurezza informatica e commercio digitale. Nei prossimi cinque anni il mondo del lavoro richiederà sempre più competenze inerenti pensiero critico, capacità analitiche, problem solving, autogestione, oltre che capacità di lavorare in team e competenze tecniche. Dal report del World Economic Forum emerge la necessità di riqualificare la forza lavoro esistente e di attivarsi per formare la forza lavoro del futuro. Infine, sarà necessaria la collaborazione tra pubblico e privato per affrontare le problematiche del settore produttivo a seguito delle trasformazioni tecnologiche ed economiche in atto [18].

1.3 Panoramica sulla manutenzione predittiva

Come descritto nei precedenti capitoli, nel contesto dell'Industria 4.0 e dell'Internet of Things, si colloca la *manutenzione predittiva*. Nella seconda parte del presente elaborato si descriverà un'applicazione pratica della stessa, analizzando i passaggi da compiere per la progettazione e implementazione dell'intero sistema nell'ambito di un macchinario industriale per lo stampaggio plastico a iniezione. Uno dei principali problemi per le aziende produttive riguarda il calo della performance e il degrado delle risorse da esse utilizzate. Tale problematica porta ad una generale diminuzione dell'efficienza dell'intero processo produttivo, nel quale possono presentarsi dei guasti. Con il termine *guasto* si indica una cessazione della capacità di un componente a eseguire una determinata funzione richiesta. La *manutenzione* ha quindi due scopi principali: riparare i guasti e impedire la loro insorgenza. Solitamente, la comparsa del guasto causa una riduzione delle prestazioni del macchinario, in base alla gravità del guasto tale riduzione può essere istantanea e totale oppure dilazionata nel tempo e parziale. Dal momento in cui il guasto si manifesta e il momento nel quale viene riparato, spesso trascorre un intervallo di tempo chiamato *ritardo amministrativo*. Tale ritardo è dovuto principalmente al fatto che gli operatori che lavorano a stretto contatto con il macchinario devono innanzitutto

venire a conoscenza della presenza del guasto. Come si può vedere nella figura 1.7, dopo che avviene un guasto vi sono diversi tempi di riparazione prima che si raggiunga nuovamente un buon livello di funzionamento del macchinario [19].

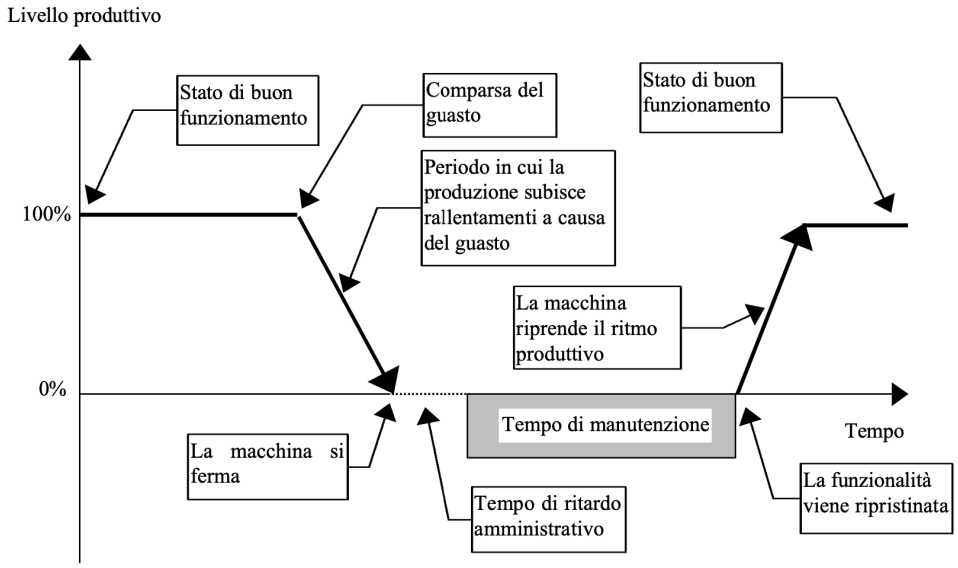

Figura 1.7: Grafico del processo di gestione di un guasto

In un primo momento il macchinario continua a funzionare, dando luogo a processi rallentati o difettosi, finché non si ferma. Successivamente l'operatore si accorge del problema e segnala il guasto agli addetti alla manutenzione che si occuperanno effettivamente della riparazione. In questa fase possono subentrare ulteriori ritardi relativi alla disponibilità dei manutentori, alla diagnostica del problema, alla disponibilità di eventuali pezzi di ricambio e a una fase di testing. Dopo aver risolto il guasto il macchinario torna gradualmente nel suo stato di funzionamento ottimale. Individuare in anticipo un possibile guasto permette quindi un notevole risparmio di tempo, risorse e costi in modo che l'intero processo produttivo possa funzionare correttamente e senza interruzioni. Per cercare di ridurre tali inefficienze nel processo produttivo, esistono diverse tipologie di manutenzione che presentano livelli crescenti di complessità e precisione. Tali tipologie di manutenzione sono riassunte nella figura 1.8 e descritte di seguito [20]:

- **Manutenzione correttiva:** è la strategia di manutenzione più semplice e tradizionale, consiste nel riparare il guasto solo dopo che esso si è verificato. Questo porta a perdita di ricavi dovuti al non utilizzo del macchinario, imprevedibilità dei guasti e costi più elevati poiché guasti a componenti che si protraggono nel tempo possono a loro volta danneggiare altre componenti del macchinario.

- **Manutenzione preventiva:** è una strategia di manutenzione il cui obiettivo è quello di prevenire i guasti inserendo cicli di manutenzione da compiere ad intervalli regolari. Si basa quindi sulla sostituzione programmata di un determinato componente della macchina ancora funzionante con uno nuovo, in modo tale da prevenirne il cedimento incontrollato.

- **Manutenzione su condizione:** è una strategia di manutenzione nella quale si controllano le condizioni di usura dei componenti e successivamente si decide se cambiarli o meno, effettuando quindi una manutenzione *su condizione*. Revisioni non necessarie di macchinari possono talvolta provocare guasti indotti e un costo maggiore rispetto al permettere che il macchinario funzioni fino alla sua rottura. La strategia di monitoraggio della condizione, eseguita con verifiche periodiche, tende quindi ad individuare lo stato di un componente che potenzialmente potrebbe provocare il guasto.

Nel corso del tempo, e in misura sempre maggiore nell'ambito dell'Industria 4.0, il concetto di manutenzione si è evoluto in un nuovo approccio: la **manutenzione predittiva**, nota anche con il termine inglese di *predictive maintenance*. La manutenzione predittiva ha lo scopo di prevedere eventuali guasti, analizzando le risorse che sono correlate tra loro e il relativo comportamento passato. In questo modo si vuole determinare un momento ottimale nel quale eseguire la manutenzione, prima che avvenga effettivamente il guasto. Contrariamente alla manutenzione preventiva e alla manutenzione su condizione, l'idea alla base della manutenzione predittiva è quella di eseguire un controllo dello stato del macchinario industriale in modo tale da non interrompere il suo normale funzionamento ma da segnalarne anticipatamente ed in modo continuo il progressivo degrado di un componente per anticipare un imminente guasto [21]. L'intero processo si basa quindi su dati che vengono raccolti in real time, istante dopo istante, da dispositivi IoT come ad

esempio sensori di temperatura e pressione, successivamente memorizzati in un database. Questo approccio è in netto contrasto con quanto avviene nella manutenzione preventiva e nella manutenzione su condizione. In queste ultime, le decisioni non vengono prese sulla base dei dati raccolti dai sensori, ma ci si affida all'esperienza degli operatori o a sistemi di allarme. I dati relativi all'accadimento di un guasto sono quindi di fondamentale importanza, poiché indicano l'istante temporale e le circostanze durante le quali è avvenuto il malfunzionamento.

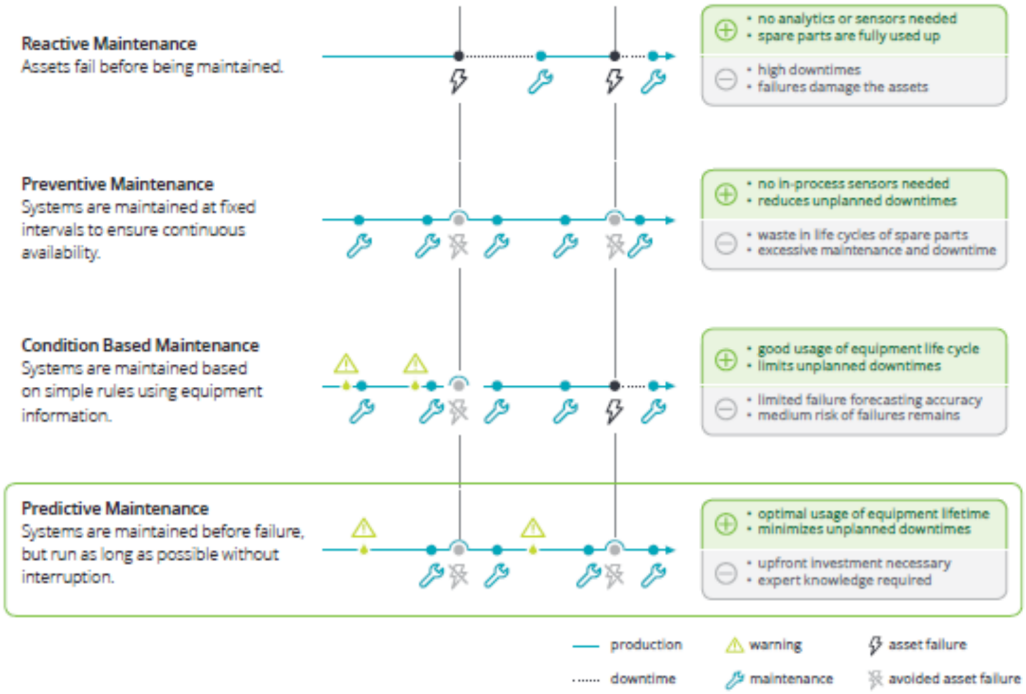

Figura 1.8: Tipologie di manutenzione di macchinari industriali

Dopo che i dati sono stati raccolti vi è una fase di elaborazione degli stessi, chiamata fase di *preprocessing*, durante la quale i dati vengono puliti e preparati per le successive elaborazioni. A partire da tali dati lavorati, tramite tecniche di intelligenza artificiale, in particolare tecniche di *Machine Learning* e *Deep Learning*, è possibile realizzare un modello in grado di eseguire la predizione sulla comparsa di un guasto. Esistono diversi algoritmi applicabili per la realizzazione di un sistema di manutenzione predittiva. Uno

Figura 1.9: Esempio di utilizzo della manutenzione predittiva

dei più usati, oltre ad essere quello utilizzato nel presente elaborato, riguarda la rilevazione delle anomalie in serie temporali di dati che possono coincidere con guasti. In fase di progettazione del sistema è anche necessario eseguire una stima del tempo rimanente prima che si verifichi un guasto dal momento in cui esso viene segnalato. Il modello realizzato viene poi valutato sulla base di determinate metriche per stabilire ad esempio l'accuratezza e la precisione del sistema. Le fasi appena descritte verranno analizzate più nel dettaglio nei capitoli successivi [22]. La manutenzione predittiva è sempre più utilizzata in molteplici settori industriali, oltre ad essere un'area di ricerca di particolare interesse visti i benefici in termini di prestazioni ed efficienza produttiva che permette di raggiungere. Da un punto di vista aziendale, la manutenzione predittiva è vista come un investimento dal quale trarre successivamente un profitto. In base alla tipologia di azienda e alla sua dimensione, devono quindi essere valutati i costi e i benefici che la manutenzione predittiva può portare secondo diversi fattori come l'installazione di sensori, la costruzione di un modello e la complessità del dominio del problema. Un altro aspetto importante da considerare è la qualità dei dati utilizzati dall'intero sistema, che devono essere il più possibile corretti e precisi, per evitare di incorrere in risultati finali non soddisfacenti [23].

Capitolo 2

Dall'Intelligenza Artificiale alla Data Science

Come è stato descritto nel capitolo precedente, l'intelligenza artificiale svolge un ruolo fondamentale nel contesto dell'Industria 4.0 e più in particolare nell'ambito della manutenzione predittiva. Per implementare il sistema di manutenzione predittiva che verrà presentato nella seconda parte dell'elaborato, sono state utilizzate diverse tecniche di intelligenza artificiale, in particolare relative al Machine Learning e al Deep Learning. L'obiettivo di questo capitolo è quello di fornire una panoramica teorica delle tecniche e degli algoritmi che sono stati utilizzati durante lo svolgimento del progetto, oltre a definire il concetto di Data Science, approccio che è stato utilizzato per considerare il dominio del problema.

Spesso vi è una certa confusione nell'utilizzo dei termini intelligenza artificiale, Machine Learning e Deep Learning, tanto da essere usati impropriamente come sinonimi. Di seguito sono riportate le relative definizioni [24] [25]:

- **Intelligenza Artificiale:** si tratta di un termine generico che indica il campo dell'informatica dedicato alla creazione di sistemi che sono in grado di risolvere problemi e di riprodurre attività proprie dell'intelligenza umana. Spesso il termine intelligenza artificiale è abbreviato in *AI*.

- **Machine Learning:** l'apprendimento automatico è una applicazione dell'intelligenza artificiale che consente alle macchine di imparare dai dati, senza che queste siano programmate in maniera esplicita. L'algoritmo riceve una serie di dati ed è capace di apprendere, modificando e migliorando le predizioni mano a mano che riceve più informazioni su ciò che sta elaborando. Per questo motivo, gli algoritmi di Machine Learning tenteranno di minimizzare gli errori e massimizzare la probabilità che le loro previsioni siano vere. Spesso il termine Machine Learning è abbreviato in *ML*.

- **Deep Learning:** l'apprendimento profondo è un sottogruppo del Machine Learning. È basato su modelli e algoritmi computazionali che imitano l'architettura delle reti neurali biologiche nel cervello umano. Tali algoritmi costruiscono e utilizzano le cosiddette reti neurali artificiali, note anche con l'abbreviazione *ANN*. Spesso il termine Deep Learning è abbreviato in *DL*.

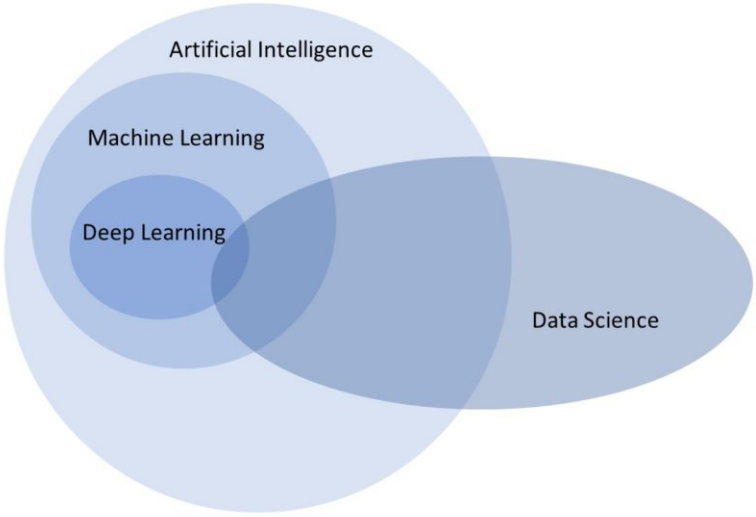

Figura 2.1: Schema sull'Intelligenza Artificiale e la Data Science

L'apprendimento automatico è quindi una disciplina scientifica che si concentra sul modo in cui i computer apprendono dai dati. Si tratta del risultato dell'incontro tra la statistica, che individua relazioni nei dati, e l'informatica. Il suo scopo è quello di costruire

modelli da set di dati che possono essere estremamente vasti. Nell'ambito del Machine Learning esistono diverse categorie di apprendimento, nei prossimi capitoli si descriveranno l'apprendimento supervisionato (in inglese *Supervised Learning*) e l'apprendimento non supervisionato (in inglese *Unsupervised Learning*), poiché sono i due approcci utilizzati nell'implementazione del progetto di manutenzione predittiva. Successivamente si descriverà il funzionamento delle reti neurali. Nella figura 2.1 è mostrata graficamente la gerarchia tra i concetti appena definiti.

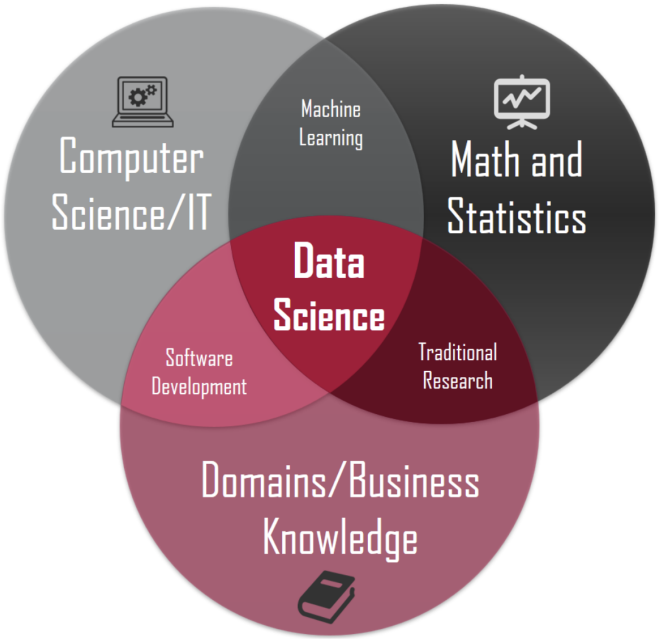

Figura 2.2: Schema sulla Data Science

La **Data Science**, che in italiano significa scienza dei dati è, un campo interdisciplinare che utilizza metodi, processi, algoritmi e sistemi scientifici per estrarre conoscenza e informazioni da insiemi di dati che possono essere strutturati o non strutturati [26]. Con Data Science si intende quindi un concetto per unire statistica, analisi dei dati e relativi metodi con lo scopo di comprendere e analizzare i fenomeni reali tramite i dati [27]. Si utilizzano tecniche e teorie tratte da molti campi nel contesto dell'informatica, della matematica, della statistica, della conoscenza del dominio e della scienza dell'informazio-

ne, come mostrato graficamente nella figura 2.2. Jim Gray, vincitore del *Turing Award* ha immaginato la scienza dei dati come un quarto paradigma della scienza: empirica, teorica, computazionale e ora basata sui dati. Inoltre ha affermato che [28]:

> *"Tutto ciò che riguarda la scienza sta cambiando a causa dell'impatto delle tecnologie dell'informazione e del diluvio di dati."*

2.1 Apprendimento supervisionato

Nell'apprendimento supervisionato l'insieme di dati da utilizzare viene diviso in tre parti: *training set* (la maggior parte del dataset), *validation set* e *test set* (parti più piccole del dataset), come mostrato nella figura 2.3. Innanzitutto viene fornito all'algoritmo un dataset di esempio, cioè il training set, nel quale i dati sono etichettati. Questo significa che gli esempi sono composti da una coppia di dati contenenti il dato originale e il risultato atteso, cioè la variabile di risposta. Pensando al progetto di manutenzione predittiva, questo significa fornire all'algoritmo un dataset composto da produzioni del macchinario industriale (che includono le caratteristiche ad esse associate, cioè i valori letti da ogni sensore) in cui l'esito di ogni produzione (corretta o con guasto) è noto. Il compito dell'algoritmo di Machine Learning è trovare la funzione che modelli la relazione tra i due, in modo da essere in grado di fare previsioni su nuovi dati in cui l'esito non è noto. Più formalmente, questo tipo di metodi servono ad individuare una funzione che predica la variabile di risposta y da un vettore di caratteristiche x che contiene n variabili in input, in modo tale che $f(x)=y$. Nel caso in cui la variabile di risposta sia di tipo categorico (come nel caso di una produzione che può essere corretta o con guasto) si parla di *classificazione*, invece nel caso di variabili di risposta continue si parla di *regressione*. Esistono diversi algoritmi di apprendimento automatico per creare funzioni che eseguano classificazioni o regressioni, ad esempio *Support Vector Machine*, *Naive Bayes*, *Logistic Regression*, *Decision Trees*, *K-Nearest Neighbors* e *Random Forest*. In seguito, il modello generato durante la fase di training viene validato sul validation set. Questo permette di selezionare il modello più performante, per esempio in termini di accuratezza. L'ultimo passaggio consiste nel valutare la performance del modello selezionato su un nuovo set di dati, il test set [29]. La suddivisione del dataset originale appena descritta è necessaria

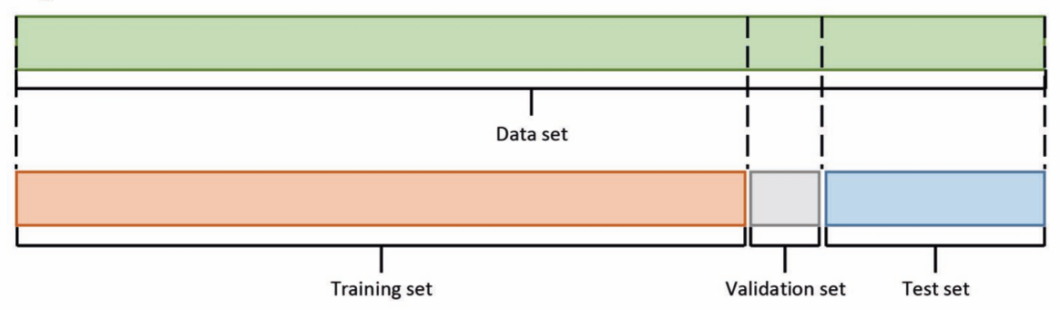

Figura 2.3: Divisione del dataset in training set, validation set e test set

per garantire di non incorrere nel cosiddetto *overfitting*, con tale termine si indica la situazione in cui il modello generato risulta essere troppo sensibile alle caratteristiche particolari del training set piuttosto che alle caratteristiche generali del problema che si sta analizzando. Nel caso in cui si presentasse l'overfitting, il modello avrebbe performance di predizione altissime nel dataset di partenza ma la sua performance risulterebbe estremamente inferiore su dati nuovi. La rappresentazione grafica del concetto appena descritto è mostrata nella figura 2.4, nella quale si evidenzia la differenza tra *overfitting*, *underfitting* e *appropriate-fitting* [30]. L'obiettivo è quello di trovare un compromesso per ottenere un modello che non si adatti eccessivamente al training set ma che non sia nemmeno troppo semplice, come in caso di underfitting [31].

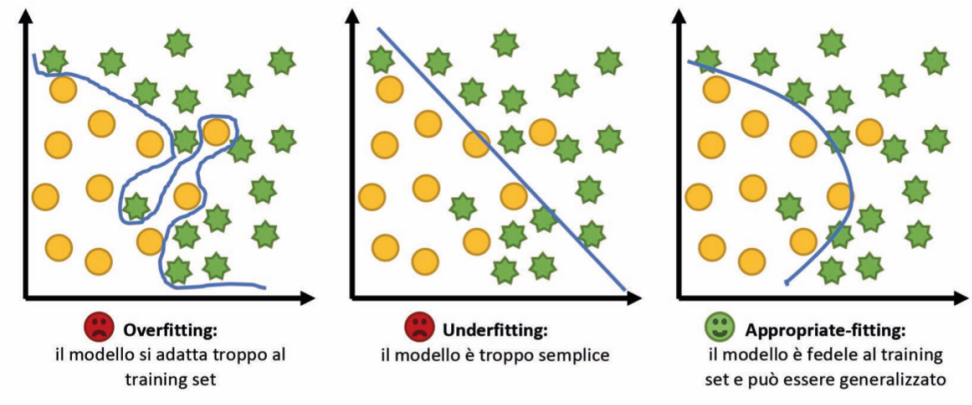

Figura 2.4: Differenza tra overfitting, underfitting e appropriate-fitting

2.1.1 Support Vector Machine

Una delle tecniche di Machine Learning utilizzate durante la realizzazione del progetto di manutenzione predittiva che sfrutta l'apprendimento supervisionato è Support Vector Machine, in italiano *macchina a vettori di supporto* e spesso abbreviato con *SVM*. Si tratta di un algoritmo utilizzato nella risoluzione di problemi sia di classificazione che di regressione, il quale risulta particolarmente efficace in caso di classificazione binaria. SVM si basa sull'idea di trovare un iperpiano che divida al meglio un dataset in due classi, nel caso di classificazione binaria. Di seguito sono descritti alcuni dei concetti su cui si basa l'algoritmo:

- **Iperpiano:** nel caso di classificazione con solo due dimensioni spaziali (x e y), un iperpiano è rappresentato da una linea che separa e classifica un insieme di dati. Nel caso di tre dimensioni viene invece rappresentato da un piano. Mentre nel caso in cui siano presenti più di tre dimensioni si definisce genericamente *iperpiano*.

- **Vettori di supporto:** si tratta dei punti che sono più vicini all'iperpiano. Tali punti dipendono dal dataset che si utilizza e in base a come si modificano influiscono sulla posizione dell'iperpiano.

- **Margine:** è la distanza tra i vettori di supporto di due classi differenti più vicini all'iperpiano. Al centro di tale distanza si posiziona l'iperpiano stesso.

Lo scopo di SVM è quello di individuare un iperpiano linearmente separabile, cioè un limite di decisione che separa i valori di una classe dall'altro. Se ne esiste più di uno, cerca quello che ha il margine maggiore rispetto ai vettori di supporto. Se tale iperpiano non esiste SVM utilizza una mappatura non lineare per trasformare i dati di allenamento in una dimensione superiore. L'iperpiano ottimale può essere definito come un prodotto scalare multidimensionale dove w è il vettore di peso, x è il vettore di caratteristiche di input e *w0* è il bias:

$$w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = 0$$

Nel caso di due dimensioni e di classificazione binaria, i punti che si trovano sopra l'iperpiano e che appartengono a una classe, soddisfano la prima condizione. Mentre i

punti che si trovano sotto l'iperpiano e che appartengono all'altra classe, soddisfano la seconda condizione:

$$w_0 + w_1 x_1 + w_2 x_2 \geq +1, \text{ per } y = +1$$

$$w_0 + w_1 x_1 + w_2 x_2 \leq -1, \text{ per } y = -1$$

Dove y può assumere solo il valore $+1$ o -1, eseguendo così la classificazione. In questo modo si costruiscono i confini del margine e i dati che ricadono esattamente su tali confini non sono altro che i vettori di supporto.

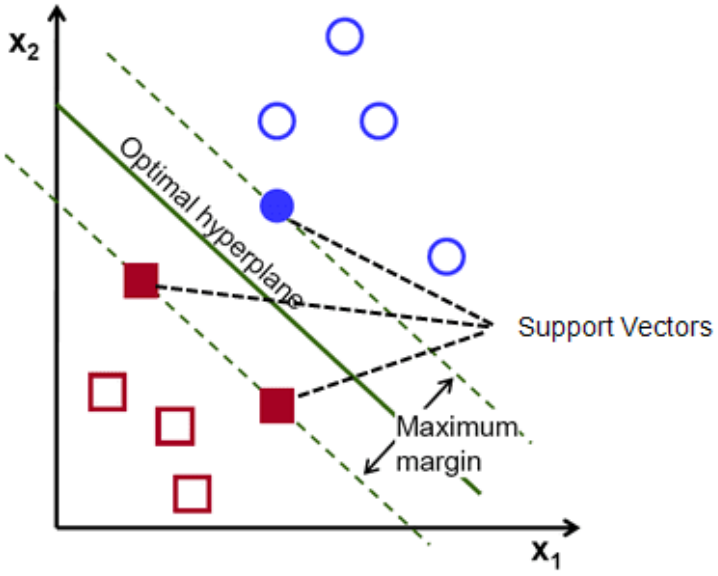

Figura 2.5: Rappresentazione dell'iperpiano con Support Vector Machine

Come accennato precedentemente, quando non esiste un limite di decisione lineare è possibile usare modelli non lineari di dimensioni superiori tramite il *metodo del kernel*. In questo caso gli algoritmi SVM utilizzano un insieme di funzioni matematiche definite come kernel. L'obiettivo è quello di trasformare i dati in input nella forma richiesta quando non è possibile determinare un iperpiano che sia linearmente separabile, come avviene nella maggior parte dei casi. Il kernel può essere definito come:

$$K(x, y) \ = \ < f(x), f(y) >$$

In questo caso K è la funzione del kernel mentre x e y sono vettori di input a dimensione n. f viene usato per mappare l'input dallo spazio n dimensionale allo spazio m dimensionale, di livello superiore rispetto ad n. Infine $<x,\ y>$ indica il prodotto scalare. È importante considerare che il metodo del kernel non fa parte dell'algoritmo Support Vector Machine, il cui scopo è di trovare il migliore iperpiano, infatti può essere usato anche con altri classificatori come *Logistic Regression*. La scelta della funzione del kernel è uno dei fattori che incide sulle prestazioni di un modello SVM. Solitamente si decide quale funzione utilizzare in base ad un determinato problema di classificazione dopo una serie di test. Nel progetto di manutenzione predittiva che verrà descritto nella seconda parte del presente elaborato si è scelto di utilizzare la funzione di kernel *RBF*, definita come:

$$f(x, y) \ = \ exp(-gamma \cdot ||x - y||^2)$$

Tale tipologia di kernel *Radial Basis Function* è noto anche con il nome di kernel *Gaussiano* e presenta un limite decisionale più complesso. Questo tipo di kernel utilizza il parametro *gamma* (γ), un piccolo valore di tale parametro fa in modo che il modello si comporti come un SVM lineare. Al contrario un grande valore di gamma rende il modello maggiormente influenzato dagli esempi dei vettori di supporto. Altre funzioni kernel sono ad esempio la *funzione lineare*, la *funzione polinomiale* e la *funzione sigmoide* [32].

2.2 Apprendimento non supervisionato

A differenza dell'apprendimento supervisionato, gli approcci non supervisionati sono utilizzati quando le etichette delle variabili in input non sono note a priori, di conseguenza tali metodi tentano di riconoscere relazioni e pattern solo dalle caratteristiche dei dati e senza utilizzare una categorizzazione come visto per gli algoritmi di Supervised Learning. L'apprendimento non supervisionato ha alcuni obiettivi: comprendere la distribuzione dei dati, raggrupparli sulla base di caratteristiche simili o ridurre le loro dimensioni in modo da costruirne una versione più sintetica ed efficace. Utilizzando i metodi di Unsupervised Learning può non essere semplice misurare quanto l'algoritmo sia accurato, poiché le prestazioni sono spesso soggettive e specifiche al dominio del problema. Gli algoritmi di apprendimento non supervisionato includono ad esempio il

clustering e le regole di associazione. Il clustering consiste nel raggruppare i dati in modo che quelli appartenenti allo stesso cluster (cioè allo stesso gruppo) condividano caratteristiche simili, mentre i dati che fanno parte di cluster differenti siano diversi secondo determinate metriche. Esistono molteplici approcci che utilizzano la tecnica del clustering, ad esempio *K-Means*, *DBSCAN* e *Agglomerative Clustering*, i quali sono stati utilizzati per implementare il progetto di manutenzione predittiva e verranno descritti nei prossimi capitoli. L'apprendimento non supervisionato viene spesso utilizzato per eseguire il preprocessing dei dati, comprimendoli prima di fornirli ad altri algoritmi di apprendimento supervisionato o a reti neurali come dati di input [33].

2.2.1 K-Means

K-Means è un algoritmo di apprendimento non supervisionato il cui scopo è quello di trovare un numero fisso di cluster all'interno di un dataset. Come descritto precedentemente, i cluster sono i gruppi nei quali vengono divisi i dati in base alla loro somiglianza, in K-Means il numero dei cluster viene scelti a priori, prima dell'esecuzione dell'algoritmo stesso. Ogni cluster raggruppa un particolare insieme di dati, che vengono chiamati *data points*. Per ogni cluster viene definito un *centroide*, cioè un punto che si trova al centro di un determinato cluster. K-Means è un algoritmo iterativo, questo significa che ripete le seguenti fasi:

- **Inizializzazione:** nella prima fase si definiscono i parametri di input per eseguire l'algoritmo. In particolare si sceglie il dataset e il numero di centroidi iniziali da utilizzare che vengono disposti casualmente. Scegliendo il numero di centroidi si scelgono i cluster di cui il dataset sarà composto, cioè i raggruppamenti che si vogliono eseguire e successivamente visualizzare.

- **Assegnazione del cluster:** nella seconda fase l'algoritmo analizza ogni data point e lo assegna al centroide più vicino, quindi ad un determinato cluster. Per eseguire questa operazione viene calcolata la distanza euclidea tra ogni data point e ogni centroide.

- **Aggiornamento della posizione del centroide:** dopo la seconda fase è probabile che si siano formati nuovi cluster, poiché a quelli precedenti si sono assegnati

o tolti alcuni data points. Di conseguenza viene ricalcolato il punto esatto del centroide, modificandone la posizione. Tale posizione è la media di tutti i data points che sono stati assegnati al nuovo cluster.

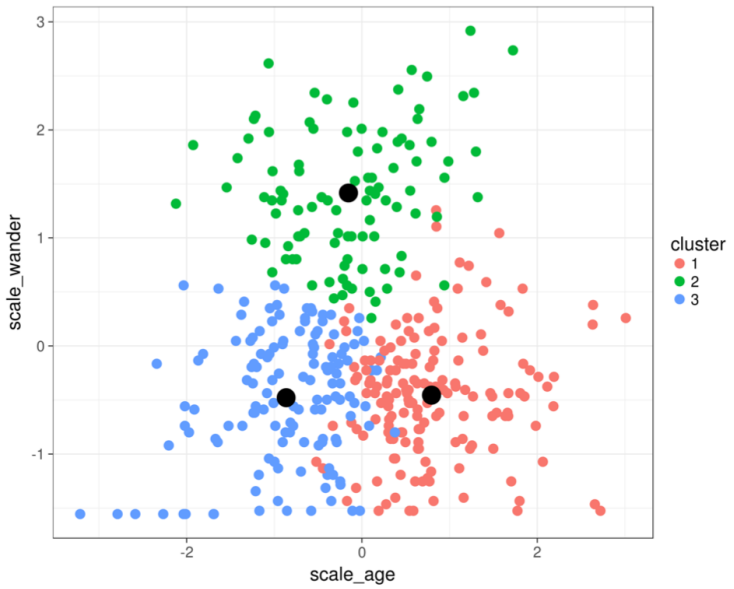

Figura 2.6: Clustering con l'algoritmo K-Means

L'algoritmo K-Means ripeterà la seconda e la terza fase finchè i centroidi non si modificano, cioè finchè non si raggiungerà un punto di convergenza tale per cui non sono più necessarie modifiche ai cluster. In questo caso viene raggiunta la condizione di stop, ad esempio quando non sono più presenti data points che cambiano cluster, quando la somma delle distanze è ridotta al minimo, oppure quando si raggiunge un numero massimo di iterazioni eseguite. K-Means è un algoritmo che ha tempi di esecuzione rapidi, adatto nel caso in cui sia noto il numero di cluster a priori ed è possibile ottenere gruppi distinti dal dataset [34].

2.2.2 Agglomerative Clustering

L'algoritmo di Agglomerative Clustering è un approccio che fa parte del cosiddetto *Hierarchical Clustering*, in italiano *clustering gerarchico*. Si tratta di un approccio di

clustering il cui scopo è quello di costruire una gerarchia tra i cluster. Come accennato, oltre ad Agglomerative Clustering, anche il *Divisive Clustering* fa parte degli approcci di Hierarchical Clustering. Il clustering agglomerativo adotta un approccio detto *bottom up*, cioè dal basso verso l'alto, nel quale si inizia inserendo ciascun elemento in un cluster diverso e successivamente si procede accorpando gradualmente coppie di cluster ampliando la soglia di similarità. L'esecuzione termina quando tutti gli oggetti sono accorpati in un unico cluster, nel quale tutti gli elementi sono considerati simili. Al contrario, il clustering divisivo adotta un approccio detto *top down*, cioè dall'alto verso il basso, nel quale tutti gli elementi inizialmente si trovano in un solo cluster che ricorsivamente viene suddiviso in cluster più piccoli.

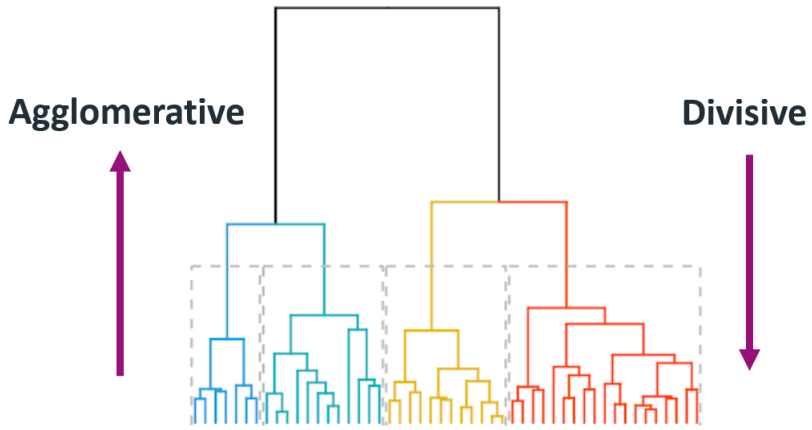

Figura 2.7: Agglomerative Clustering e Divisive Clustering

L'output del Hierarchical Clustering viene rappresentato da un dendogramma. Il modo in cui si decide quali cluster devono essere combinati, nel caso di Agglomerative Clustering, o quale cluster deve essere suddiviso, nel caso di Divisive Clustering, viene definito da una misura di dissimilarità tra i cluster. Questo si ottiene utilizzando una metrica appropriata, la quale quantifica la distanza tra coppie di elementi, oltre a un criterio di collegamento che ha il compito di definire la dissimilarità presente tra due cluster come funzione di distanze tra gli elementi nei cluster. Alcuni esempi di criteri di collegamento sono il *complete linkage*, il *minimum linkage* o l'*average linkage*. Esistono diverse me-

triche che possono essere scelte, tale scelta influenzerà la forma dei cluster poiché due elementi potranno essere più lontani o più vicini tra loro a seconda della metrica scelta. Alcuni esempi di metriche sono la *distanza euclidea*, la *distanza di Manhattan* e la *distanza di Hamming*. L'algoritmo di Agglomerative Clustering è accurato ma è poco efficiente dal punto di vista computazionale, oltre a non prevedere una riassegnazione degli elementi ai cluster durante l'esecuzione [34].

2.2.3 DBSCAN

L'algoritmo DBSCAN, acronimo di *Density-Based Spatial Clustering of Applications with Noise* è un metodo di clustering molto utilizzato e basato sulla densità, poiché il suo scopo è quello di collegare insiemi di punti che abbiano una densità sufficientemente alta. Inoltre è in grado individuare cluster che presentano forme arbitrarie. Al contrario delle tecniche K-Means e Agglomerative Clustering viste nei capitoli precedenti, l'algoritmo DBSCAN non necessita come input iniziale del numero dei cluster. I due input di cui ha bisogno sono il parametro *eps* (ε), e il parametro *minpoints*. Eps definisce il raggio della distanza, mentre minpoints è una soglia che indica il numero di vicini di ciascuna osservazione. Dato un insieme di punti che si vogliono raggruppare in cluster, innanzitutto essi sono classificati dall'algoritmo DBSCAN in una delle seguenti tre categorie:

- **Core point:** un campione p che ha almeno un certo numero minpoints di punti vicini, entro un intervallo determinato dal parametro eps. Tali punti si dicono direttamente raggiungibili dal punto p.
- **Border point:** un campione che ha un numero inferiore di minpoints come vicini, comunque collegati (direttamente o indirettamente) a un core point.
- **Outlier:** un campione che ha un numero inferiore di minpoints come vicini e non è collegato ad alcun core point.

Gli altri concetti principali su cui si basa l'algoritmo DBSCAN sono descritti di seguito:

- **Direttamente raggiungibile in densità:** dati due punti q e p, il punto q è detto direttamente raggiungibile da p se non sono lontani più di una certa distanza eps.

Come descritto in precedenza, il parametro eps è impostato dall'utente e indica l'area entro cui cercare punti vicini.

- **Raggiungibile in densità:** dati due punti q e p, q si dice raggiungibile in densità p se esiste una sequenza $p1...pn$ di punti con $p1{=}p$ e $pn{=}q$ nella quale ognuno di essi è direttamente raggiungibile dal suo predecessore.
- **Densamente connesso:** dati due punti q e p, si dicono connessi in densità se esiste un punto o tale che sia o-p che o-q siano raggiungibili in densità.

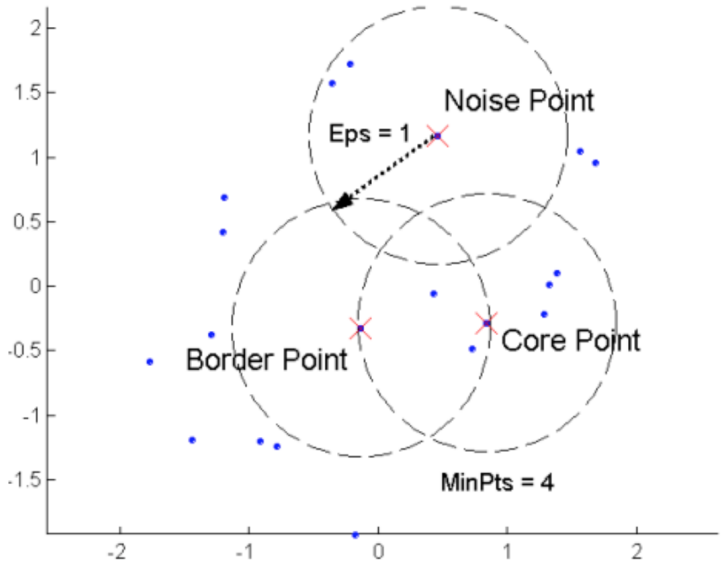

Figura 2.8: Clustering con l'algoritmo DBSCAN

Di conseguenza un cluster è un insieme in cui tutti i punti al suo interno sono connessi in densità. Inoltre, se un punto è connesso in densità ad un altro punto del cluster allora anch'esso fa parte del cluster. DBSCAN inizia la sua esecuzione selezionando casualmente un punto p non ancora visitato e ne calcola il suo vicinato in base al parametro eps. Se tale vicinato contiene un numero sufficiente di punti in base al parametro minpoints, viene creato un nuovo cluster. Se ciò non avviene il punto viene etichettato come outlier e successivamente potrebbe essere ritrovato in un altro vicinato sufficientemente grande

riconducibile ad un punto differente, entrando così a fare parte di un diverso cluster. Se un punto viene associato ad un cluster di conseguenza anche tutti i punti del suo vicinato, calcolato in base ad eps, entreranno a fare parte del medesimo cluster, lo stesso avviene anche per i loro vicinati. Tale processo continua finché il cluster è completato e sono stati visitati tutti gli altri punti [34].

2.3 Reti neurali artificiali

Le reti neurali artificiali rientrano nelle tecniche del Deep Learning e sono utilizzabili con i due approcci di apprendimento appena descritti. Sono modelli di classificazione il cui funzionamento è ispirato alla struttura e al comportamento dei neuroni del cervello umano e alle relative sinapsi. Tali neuroni sono in grado di trasmettere un segnale elettrico verso altri neuroni, in questo modo si creano delle correnti la cui somma, nel caso in cui superi una determinata soglia, si trasforma in un impulso chiamato *spike*. Nella figura 2.9 è mostrato un confronto tra un neurone biologico e uno artificiale [35].

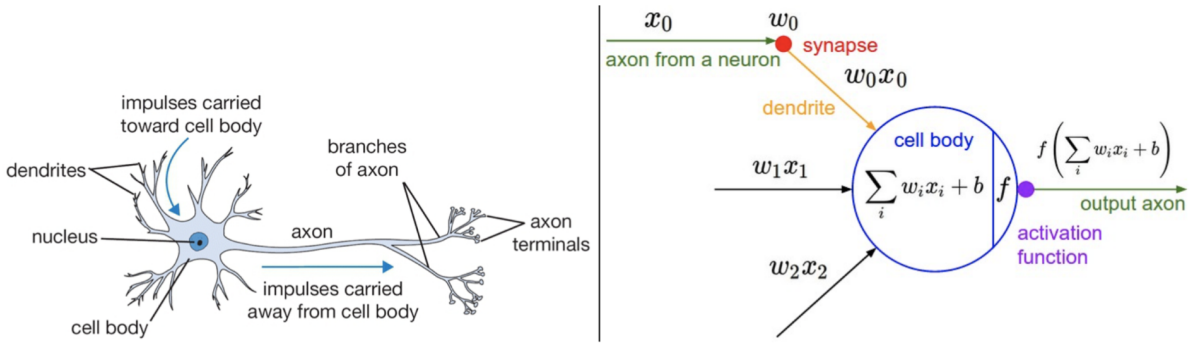

Figura 2.9: Confronto tra neurone biologico e neurone artificiale

Come avviene nella controparte biologica, il *neurone* è l'unità base di una rete neurale artificiale. Ogni neurone è formato da uno o più input, da un elemento che somma gli input ad ogni peso e da una funzione di attivazione che ha il compito di modellare il segnale in uscita. Infine, il neurone è collegato a uno o più output. Quanto appena

descritto può essere riassunto dalla seguente formula, dove x_i sono gli input, w_i sono i pesi ad essi associati, b è il bias (una soglia usata per modificare il comportamento della funzione di attivazione) e infine f rappresenta la funzione di attivazione:

$$y = f(\sum_{i=1}^{n} x_i \cdot w_i + b)$$

Una rete neurale dispone di una struttura stratificata e connessa nella quale ogni *layer*, cioè ogni strato, è formato da una serie di neuroni che ricevono in input l'informazione prodotta in output dallo strato precedente e a loro volta generano un output per il layer successivo. Come è mostrato nella figura 2.10, è possibile definire tre diversi tipi di layer: *input layer*, *output layer* e *hidden layers*. Questi ultimi sono una serie di strati nascosti che contengono un numero di nodi da definire, tali layers vengono implementati in base a come si vuole strutturare la rete neurale artificiale.

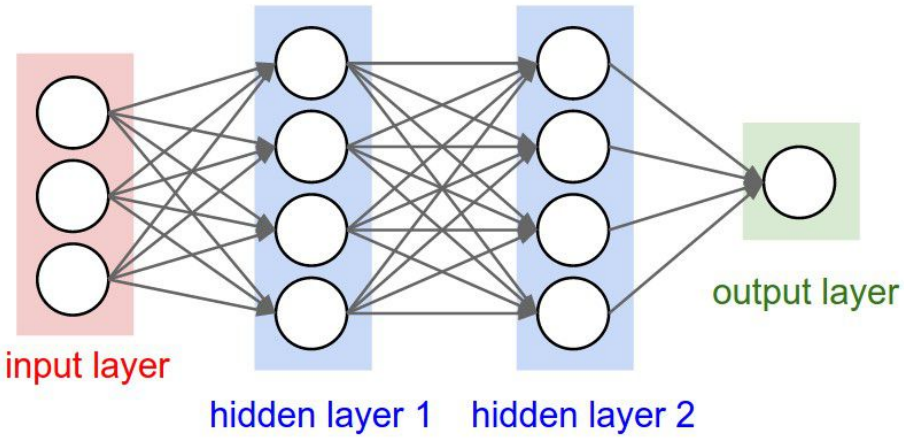

Figura 2.10: Topologia di una rete neurale artificiale

Come descritto precedentemente, ad ogni neurone viene applicata una funzione di attivazione, in questo modo si garantisce la non linearità. Questo significa che l'output non può essere riprodotto da una combinazione lineare degli input. Senza una funzione di attivazione non lineare, indipendentemente dal numero degli hidden layers, la rete si comporterebbe proprio come un *percettrone*, cioè la più semplice rete neurale a strato

singolo. Esistono diverse funzioni di attivazione che possono essere utilizzate, ad esempio la *funzione sigmoide*, la *funzione gradino* e la *funzione ReLU*. Nel progetto di manutenzione predittiva descritto nel presente elaborato si è utilizzata proprio quest'ultima. La funzione ReLU (acronimo di *Rectified Linear Unit*) viene definita come la parte positiva del suo argomento ed è mostrata nella seguente formula, dove x è l'input del neurone:

$$f(x) = x^+ = max(0, x)$$

Presenta diversi vantaggi rispetto ad esempio alla funzione sigmoide, come una maggiore efficienza e una migliore propagazione del gradiente, proprio per questi motivi risulta essere più utilizzata [36]. La forma della funzione ReLU è mostrata nella figura 2.11 [37].

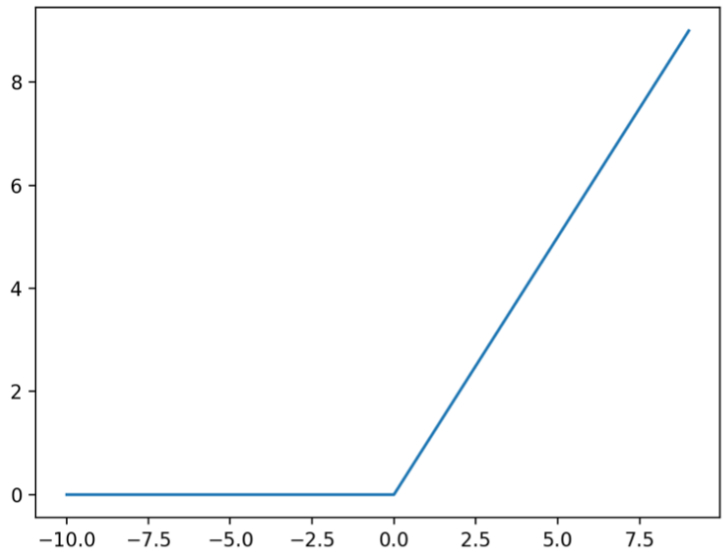

Figura 2.11: Grafico della funzione ReLU

Il processo di apprendimento della rete neurale, cioè la fase di training, consiste nella modifica dei pesi per riuscire a produrre un output coerente con gli esempi che sono stati forniti in input. Il processo di apprendimento si compone di due fasi [38]:

- **Forward propagation:** in questa prima fase i dati passano dal layer di input al layer di output attraversando gli hidden layers. Si esegue il prodotto scalare di

ogni input per il rispettivo peso e si applica la funzione di attivazione associata al layer successivo di arrivo. Tale processo viene iterato fino ad arrivare al layer di output. Dopo la prima forward propagation l'output della rete neurale non risulterà soddisfacente. L'obiettivo è quello di trovare dei valori ottimali per i pesi per fare in modo che con qualsiasi input la rete riesca a produrre un output corretto, cioè a classificare correttamente i dati. Per raggiungere tale obiettivo si vuole minimizzare l'errore, procedendo con la backward propagation.

- **Backward propagation:** l'output della forward propagation viene confrontato con il risultato atteso, tale confronto è possibile utilizzando la *loss function* che fornisce una misura dell'errore commesso dal modello. L'errore viene propagato all'indietro in tutti i nodi della rete aggiornandone i relativi pesi in modo da minimizzare la funzione di perdita. Con questo procedimento è possibile trovare i valori ottimali per tutti i pesi in modo da ottenere il minimo errore. Per raggiungere questo risultato l'algoritmo sposta i pesi nella direzione verso il punto di minimo della funzione di perdita. Tale direzione è data dall'opposto del *gradiente* e questo procedimento è chiamato *gradient descent*, cioè *discesa del gradiente*. Il gradiente è un vettore nel quale ogni suo componente è una derivata parziale rispetto a una specifica variabile. Poiché si può vedere una rete neurale come una grande funzione composta, si procede calcolando il gradiente della loss function rispetto a ogni singolo peso, utilizzando la regola della derivata della funzione composta, la cosiddetta *chain rule*. Per modificare i pesi si procede ricorsivamente, sottraendo al vettore w il gradiente moltiplicato per il parametro *alpha*, chiamato *learning rate*, cioè il tasso di apprendimento:

$$w_{i+1} \;=\; w_i - \alpha \cdot \nabla w_i$$

Se il valore di alpha è troppo piccolo, la convergenza potrebbe essere troppo lenta. Se al contrario è troppo grande, il metodo potrebbe non convergere mai, rischiando di superare il punto di minimo. Sono presenti diverse varianti dell'algoritmo di discesa del gradiente, nelle reti neurali utilizzate per implementare il progetto di manutenzione predittiva si è utilizzata la discesa del gradiente detta *mini-batch*. Tale tecnica velocizza il processo di apprendimento raggiungendo prima la conver-

genza. Questo avviene stimando il gradiente su un sottoinsieme di elementi invece che sull'intero training set. Ad ogni nuovo passaggio si utilizzerà un diverso sottoinsieme di elementi in modo da utilizzarli tutti al procedere della fase di training. Altre tecniche di discesa del gradiente sono la *batch gradient descend*, che esegue un aggiornamento solo dopo che tutte le osservazioni sono state valutate. Mentre la *stochastic gradient descend* esegue un aggiornamento per ogni osservazione [39].

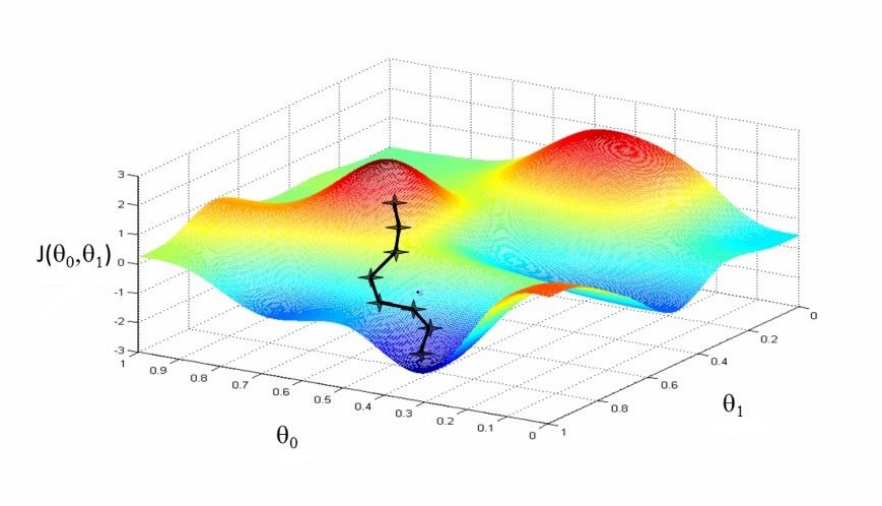

Figura 2.12: Rappresentazione della discesa del gradiente

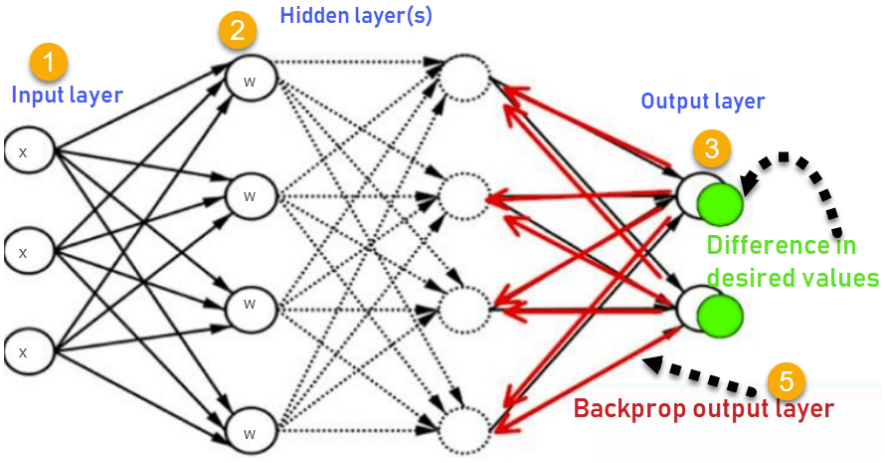

Figura 2.13: Forward propagation e backward propagation

Il procedimento appena descritto, quindi la forward propagation seguita dalla backward propagation, è chiamato *iterazione* e viene eseguito su un certo numero di esempi determinato dalla dimensione del *batch*. Con il termine *epoca* si intende invece l'insieme delle iterazioni eseguite su tutti gli esempi del dataset. Solitamente per allenare una rete neurale il processo è ripetuto un determinato numero di epoche.

2.3.1 Convolutional Neural Network

Le Convolutional Neural Network, in italiano reti neurali convoluzionali, note anche con l'acronimo *CNN*, sono un tipo di rete neurale il cui funzionamento è ispirato alla corteccia visiva animale. Sono composte sempre da un input layer, una serie di hidden layers e un output layer, ma la particolarità sta nel fatto che gli strati nascosti non devono essere necessariamente tutti completamente connessi. Ognuno di essi ha il compito di svolgere una specifica operazione e sono principalmente di quattro tipi diversi:

- **Convolutional layer:** è lo strato principale che caratterizza il funzionamento stesso della CNN, utilizza un insieme di filtri che vengono applicati ai dati in input e si modificano durante la fase di allenamento della rete. Spesso si utilizza come input un'immagine, ma un procedimento simile può essere applicato a dati di natura diversa. Ad esempio, nel sistema di manutenzione predittiva realizzato, la matrice di input è composta dai dati letti dai sensori presenti nel macchinario industriale. Tornando all'esempio dell'immagine, essa è da intendere come una matrice di pixel con una determinata altezza, larghezza e profondità. Quest'ultima è riferita ai canali dei colori *RGB*, cioè rosso, verde e blu. Ogni filtro da applicare all'immagine ha lo stesso numero di canali ma possiede una sua altezza e una sua larghezza. Durante la forward propagation ogni filtro scorre sull'immagine e viene calcolato il prodotto scalare tra la matrice di input e le componenti del filtro stesso. Tale operazione, detta *convoluzione*, produce una matrice chiamata *feature map*, con un numero di dimensioni uguale al numero dei filtri applicati. La feature map contiene quindi il risultato dell'applicazione dei filtri ad ogni pixel dell'immagine e durante la fase di training la rete impara i valori di tali filtri. Maggiore è il numero di filtri scelto e maggiore sarà il numero di *features* (cioè di caratteristiche) che

potranno essere estratte dai dati, di conseguenza la CNN riconoscerà meglio nuove immagini. In fase di costruzione della CNN bisogna specificare il numero di filtri da utilizzare, la loro dimensione (detta *kernel size*) e lo *stride*, cioè in quale misura fare scorrere il filtro sull'immagine. Infine è possibile specificare il *padding*, cioè il numero di zeri aggiunti nel bordo della matrice di input per regolare la dimensione della feature map [40].

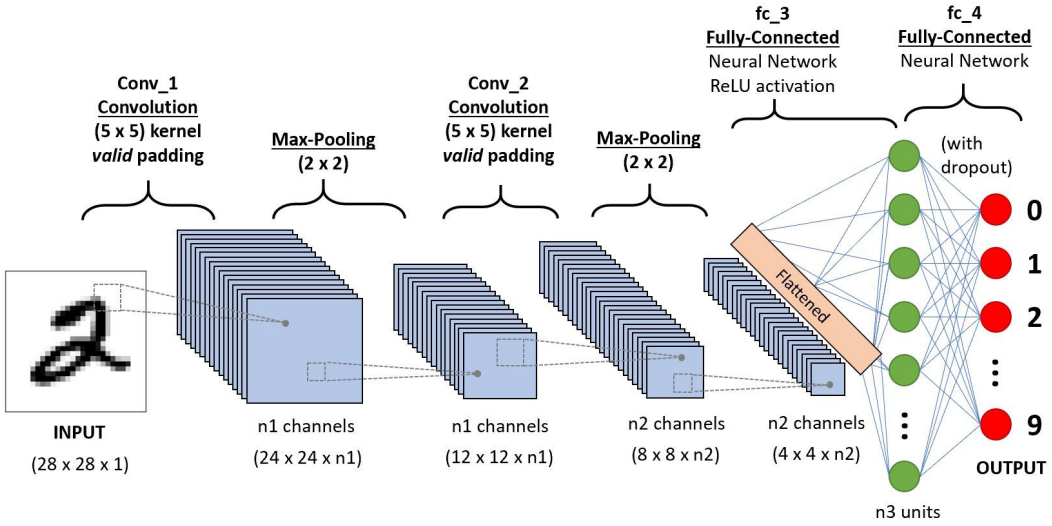

Figura 2.14: Topologia di una rete neurale convoluzionale

- **ReLU layer:** è lo strato posto alla fine di ogni convoluzione. Il suo scopo è quello di applicare una funzione non lineare alla feature map e l'output prodotto viene chiamato *rectified feature map*. La funzione ReLU applicata è quella descritta precedentemente, ovviamente possono essere applicati altri tipi di funzioni come ad esempio la funzione sigmoide.

- **Pooling layer:** è uno strato il cui scopo è quello di ridurre la dimensione delle feature map, mantenendo però le informazioni più rilevanti. Il metodo più utilizzato per compiere questa operazione è chiamato *max pooling*. Consiste nel suddividere la feature map in sottoinsiemi, mantenendo solo il massimo elemento di ogni sottoinsieme. In questo modo la rappresentazione delle feature si riduce di dimensioni, la

rete è meno influenzata da piccole modifiche dell'input e aumenta la velocità computazionale. Utilizzando l' *average pooling* invece non viene selezionato il valore massimo ma il valore medio. Altre tipologie di pooling layer sono il *global max pooling* e il *global average pooling*.

- **Fully connected layer:** è un livello che si comporta come un *percettrone multistrato*, nel quale ogni layer è completamente connesso al layer successivo. Il suo obiettivo è di classificare l'input in base alle feature map generate nei livelli precedenti, infatti solitamente è posizionato dopo una serie di layer convoluzionali e di pooling [41].

2.3.2 Autoencoder

Con Autoencoder si definisce una rete neurale artificiale cha ha l'obiettivo di trovare una codifica per il suo input in una rappresentazione compressa dello stesso. A partire da tale codifica deve riuscire a ricostruire l'input stesso. In particolare, come mostrato nella figura 2.15, un autoencoder è formato dall'unione di due sottoreti [42]:

- **Encoder:** si occupa di calcolare la funzione *z=q(x)*. Dato un input x l'encoder lo codifica in una variabile z, nota come *variabile latente*. Quest'ultima normalmente ha dimensioni ridotte rispetto a x.

- **Decoder:** si occupa di calcolare la funzione *x'=p(z)*. Data z la codifica di x prodotta dall'encoder, l'obiettivo è fare in modo che il decoder la decodifichi in maniera tale che x' sia simile ad x.

L'obiettivo della fase di training nelle reti neurali con autoencoder è quello di minimizzare l'errore quadratico medio (*MSE*, acronimo di *Mean Squared Error*) tra l'input e la ricostruzione. Si utilizza la seguente formula per input di dimensione n:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x}_i)^2$$

Le reti neurali artificiali con autoencoder non servono solo per eseguire una compressione dell'input oppure per imparare un'approssimazione della funzione identità. Partendo

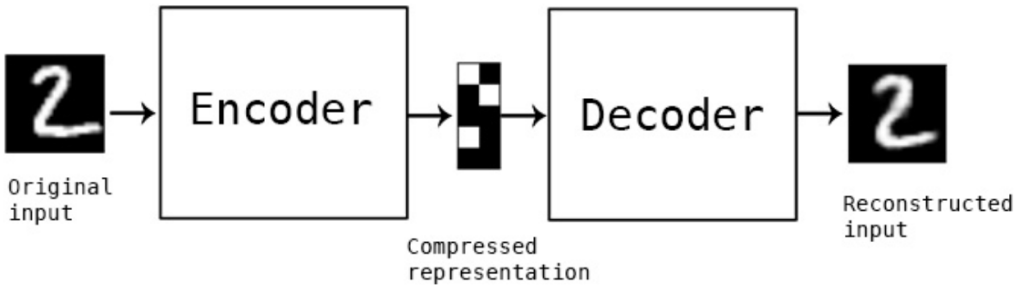

Figura 2.15: Struttura di un autoencoder

da un hidden layer di dimensioni ridotte, vi sono tecniche che permettono di forzare il modello a dare la priorità a determinate caratteristiche dei dati, dando origine a rappresentazioni diverse degli stessi [43].

Parte II

Realizzazione del progetto aziendale

"Siate affamati, siate folli."
Steve Jobs

Capitolo 3

Progettazione e modello CRISP-DM

Nella prima parte del presente elaborato si è data una panoramica sul tema dell'Industria 4.0 con particolare riferimento ad una sua applicazione, cioè la manutenzione predittiva, per poi descrivere le tecniche di intelligenza artificiale utilizzate per la sua realizzazione. L'obiettivo di questo capitolo e più in generale della seconda parte dell'elaborato, è di esporre un caso pratico di progettazione, implementazione e validazione di un sistema informatico per la manutenzione predittiva di macchinari industriali tramite tecniche di intelligenza artificiale. Ho svolto il lavoro descritto durante un tirocinio della durata di quattro mesi presso la software house *Open Data* di Funo di Argelato (Bologna), tra ottobre 2020 e febbraio 2021. Il tirocinio è stato svolto in modalità *smart working* a causa della pandemia in corso di COVID-19.

Come accennato precedentemente, l'obiettivo del progetto è stato quello di realizzare un sistema informatico per la manutenzione predittiva di macchinari industriali tramite tecniche di intelligenza artificiale. In particolare utilizzando gli algoritmi di Machine Learning e Deep Learning descritti a livello teorico nel secondo capitolo del presente elaborato. Verrà utilizzata una rilevazione delle anomalie in serie temporali di dati, che possono coincidere con guasti, il macchinario esegue nuove letture dei suoi sensori ogni secondo. Il sistema proposto rappresenta una nuova funzionalità che, con opportune modifiche e miglioramenti, potrà essere integrata all'interno di *Opera MES*, il software MES prodotto da Open Data e descritto nei capitoli successivi. L'esecuzione dell'intero siste-

ma di manutenzione predittiva dovrà avvenire tramite *cloud computing*, inoltre è stata progettata una variante per consentire l'esecuzione del sistema sfruttando un approccio di *edge computing*.

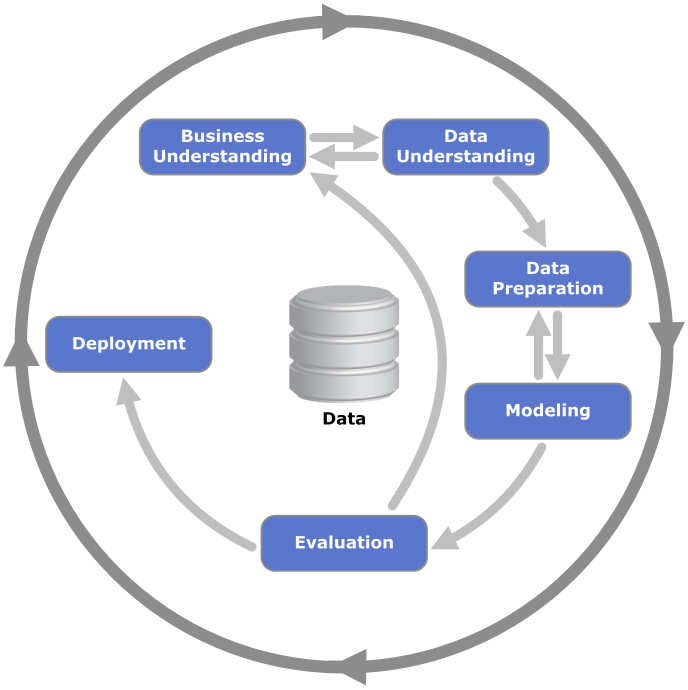

Figura 3.1: Modello di processo CRISP-DM

Durante tutto lo svolgimento del lavoro, dalla fase iniziale a quella finale, è stato seguito il modello di processo chiamato *CRISP-DM*, acronimo di *Cross-Industry Standard Process for Data Mining*. Tale modello è stato ideato nel 1996 da cinque aziende del settore e nel 1997 è diventato un progetto dell'Unione Europea nell'ambito dell'iniziativa di finanziamento *ESPRIT*. È un modello di processo open standard che descrive gli approcci comuni utilizzati nell'ambito del *Data Mining*, oltre ad essere il modello di analisi più utilizzato nel settore grazie anche al fatto che è adattabile a molti ambiti e tecnologie di utilizzo. Con il termine Data Mining si intende l'insieme delle tecniche e delle metodologie che hanno lo scopo di estrarre informazioni utili da grandi quantità di dati (come database e data warehouse), attraverso metodi automatici o semi-automatici (come Machine Learning e Deep Learning) e l'utilizzo scientifico, aziendale, industriale o operativo

delle stesse. Il Data Mining può essere considerato un sottoinsieme della Data Science, quest'ultima, come descritto nel secondo capitolo è un campo di studio più ampio che include altre discipline come informatica, statistica e conoscenza del dominio del problema.

Come mostrato nella figura 3.1, il modello CRISP-DM è composto da sei differenti fasi. La sequenza di tali fasi non è rigida ma ci si può spostare avanti e indietro tra esse se necessario. Le frecce nel diagramma di processo indicano le dipendenze più importanti e frequenti tra le fasi, mentre il cerchio esterno nel diagramma simboleggia la natura ciclica del Data Mining stesso. Un processo di Data Mining continua anche dopo che una soluzione è stata raggiunta e rilasciata, poiché quanto appreso durante il processo può innescare nuove domande aziendali più specifiche nelle successive iterazioni. Quindi ogni nuovo processo di Data Mining trae vantaggio dall'esperienza maturata nei processi precedenti [44]. Di seguito sono elencate le sei fasi che compongono il modello CRISP-DM, tali fasi verranno approfondite nei prossimi capitoli analizzando le operazioni che si sono compiute in ognuna di esse [45]:

- **Business Understanding:** questa fase iniziale si concentra sulla comprensione degli obiettivi e dei requisiti del progetto da una prospettiva aziendale, convertendo quindi questa conoscenza in una definizione del problema di Data Mining e in un piano preliminare progettato per raggiungere gli obiettivi.

- **Data Understanding:** la fase di comprensione dei dati inizia con una raccolta iniziale degli stessi e procede con le attività al fine di acquisire familiarità con i dati, identificare problemi di qualità, scoprire le prime intuizioni o rilevare sottoinsiemi interessanti per formare ipotesi di informazioni nascoste.

- **Data Preparation:** la fase di preparazione dei dati copre tutte le attività per costruire il dataset finale dai dati grezzi iniziali, necessario per le fasi successive.

- **Modeling:** in questa fase vengono selezionate e applicate varie tecniche di modellazione e i loro parametri vengono calibrati su valori ottimali, in modo tale da costruire i migliori modelli di apprendimento possibili.

- **Evaluation:** in questa fase il modello o i modelli ottenuti vengono valutati in modo più approfondito e le fasi eseguite per costruire il modello vengono riviste per essere certi che esso raggiunga correttamente gli obiettivi di business. Infine viene valutato il modello finale che è stato costruito.

- **Deployment:** la creazione del modello generalmente non è la fine del progetto. Anche se lo scopo del modello è aumentare la conoscenza dei dati, tale conoscenza acquisita dovrà essere organizzata e presentata in modo che il cliente possa effettivamente utilizzarla, di conseguenza avviene il rilascio del sistema.

3.1 Business Understanding

In questa prima fase del modello CRISP-DM viene definito il dominio del problema di Data Mining che si vuole risolvere, concentrandosi sulla comprensione degli obiettivi e dei requisiti del progetto da una prospettiva di business. Questa fase è composta dai seguenti task:

- **Determinazione degli obiettivi di business:** è necessario innanzitutto comprendere a fondo, da una prospettiva di business, ciò che il cliente vuole veramente realizzare, definendo i criteri per raggiungere l'obiettivo.

- **Valutazione della situazione:** si determinano la disponibilità delle risorse e i requisiti del progetto. Avviene la valutazione di rischi e imprevisti ed eventualmente si conduce un'analisi costi-benefici.

- **Determinazione degli obiettivi del Data Mining:** oltre a definire gli obiettivi di business, è necessario definire anche gli obiettivi da un punto di vista tecnico relativamente al processo di Data Mining.

- **Produzione del piano di progetto:** si selezionano tecnologie e strumenti da utilizzare durante il progetto e si definiscono piani dettagliati per tutte le fasi successive del modello.

Si tratta di una fase estremamente importante dalla quale dipenderà l'esito dell'intero processo. Spesso capita che i team si affrettino a superare questa fase iniziale, non

considerando che stabilire una forte comprensione aziendale è assolutamente essenziale come costruire le basi su cui si costruirà l'intero progetto. In questa fase vi è stato innanzitutto il mio inserimento all'interno del contesto aziendale, successivamente si è analizzato il dominio del problema relativo alla manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione, comprendendone il loro funzionamento e i possibili difetti ai prodotti realizzati. Si sono comprese la struttura e le funzionalità di Opera MES, il software nel quale dovrà essere integrato l'intero sistema di manutenzione predittiva, oltre ad approfondire cosa è un *Manufacturing Execution System*. Infine si è approfondita la tematica dell'analisi dei dati e il funzionamento delle tecniche di Machine Learning e Deep Learning descritte nella prima parte del presente elaborato, utilizzate nelle fasi successive del processo.

3.1.1 Panoramica su Open Data

Open Data, l'azienda nella quale ho svolto il progetto di tirocinio per tesi descritto nel presente elaborato, è una software house italiana, leader sul mercato internazionale dei sistemi MES e produttore del software Opera MES. L'azienda nasce nel 1994 come società che realizza i primi sistemi informativi online per la gestione e il controllo della fabbrica ed è oggi un produttore MES internazionale con il prodotto Opera MES, distribuito in tutto il mondo da un network di *partner* e *system integrator* che utilizzano la piattaforma per la realizzazione di fabbriche intelligenti.

Figura 3.2: Logo di Open Data

Nel 2020 Open Data è entrata a far parte del gruppo *Zucchetti*, principale azienda ICT italiana che realizza soluzioni con la tecnologia più avanzata in Europa. La continua crescita dell'azienda, sempre al passo con i nuovi trend di mercato e con l'evoluzione delle tecnologie, ha portato alla nascita di nuove funzionalità basate sull'intelligenza artificiale e l'implementazione di reti neurali artificiali. Le tecnologie e i prodotti Open Data in ambito MES ed intelligenza artificiale rendono intelligente il business delle aziende clienti e forniscono gli strumenti per eccellere nelle performance ed incrementare il business nell'era della trasformazione digitale. I prodotti sviluppati sono frutto di competenze multidisciplinari, grazie ad un team eterogeneo e propenso ad innovare. Open Data è composta da due centri di innovazione e di sviluppo software che progettano e sviluppano i prodotti, supportati a livello internazionale da un ecosistema di partner che utilizza le soluzioni per implementare i progetti presso i clienti. Open Data è presente in oltre 18 paesi in tutto il mondo (Europa, Stati Uniti d'America, centro e sud America, Cina, Russia e Brasile) con un network di oltre 40 partner che si occupano della promozione, implementazione, localizzazione e supporto dei prodotti. Inoltre possiede più di 500 clienti a livello internazionale in molteplici settori industriali e di dimensioni diverse [46].

3.1.2 Manufacturing Execution System e panoramica su Opera MES

Come descritto in precedenza, il sistema di manutenzione predittiva realizzato rappresenta una nuova funzionalità del software Opera MES. Tale software è un *Manufacturing Execution System*, spesso abbreviato con l'acronimo *MES*. Con questo termine si indica un sistema informatizzato il cui scopo principale è quello di gestire e controllare la funzione produttiva di un'azienda. La gestione coinvolge diverse aree, tra le quali l'invio degli ordini, gli avanzamenti in quantità e tempo, oltre al versamento delle risorse nel magazzino. Un'altra area fondamentale è rappresentata dal collegamento diretto ai macchinari industriali, per acquisire informazioni necessarie ad integrare l'esecuzione del processo produttivo ed elaborare dati per controllare ogni aspetto della produzione. Un MES è quindi un sistema software utilizzato per gestire in modo integrato, intelligente ed efficiente l'intero processo produttivo di un'azienda, nell'ottica dell'Industria 4.0. Questo avviene tramite collegamenti diretti ai macchinari, ad esempio con *controllori logici*

programmabili (noti con l'acronimo *PLC*) oppure con input manuali degli operatori che lavorano a stretto contatto con le macchine. Tutte le informazioni prodotte sono rese disponibili in tempo reale all'area produttiva e agli uffici. Tale meccanismo permette loro di avere costantemente una visione completa dell'intero processo produttivo, dallo stato delle risorse utilizzate all'avanzamento degli ordini, passando per i materiali impiegati. Altra importante caratteristica riguarda la possibilità di inviare le informazioni prodotte dal software MES al sistema gestionale ERP già utilizzato dall'azienda.

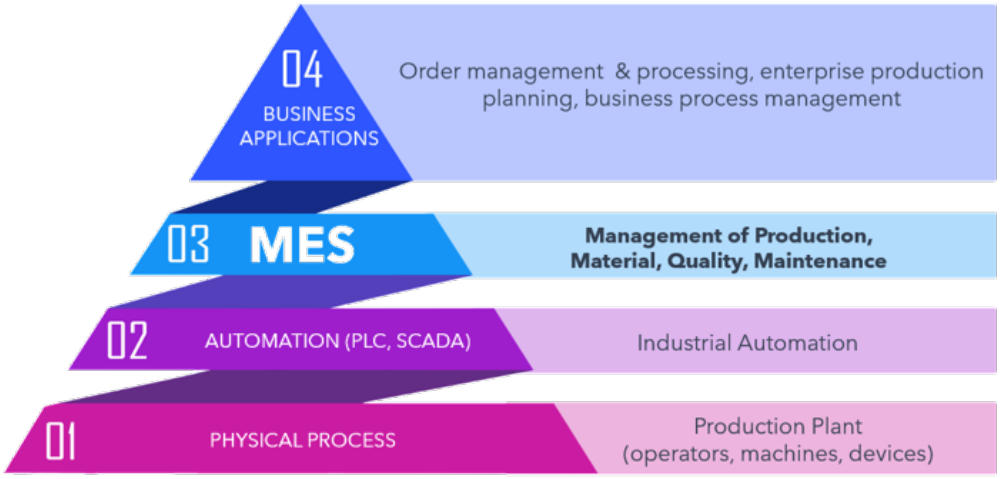

Figura 3.3: Gerarchia dei livelli nella fabbrica digitale

Tramite un software MES è quindi possibile allineare la gestione aziendale e la produzione, in modo da evitare differenze tra il livello produttivo pianificato e quello realmente attivo. Grazie al MES è possibile realizzare il prodotto finito rispettando le richieste del reparto vendite, questo avviene aumentando l'efficienza di ogni fase della filiera produttiva, includendo operatori, macchinari industriali e risorse utilizzate. Tra le principali funzionalità e caratteristiche di un software MES figurano il monitoraggio in tempo reale di ogni attività svolta, includendo gli operatori e i macchinari industriali coinvolti in tali attività, il riconoscimento di eventuali guasti e malfunzionamenti durante la produzione, il controllo delle specifiche di prodotto, della qualità, dei tempi di esecuzione e dei materiali utilizzati [47].

Opera MES è il *core business* di Open Data. Diversi team dedicati si occupano della progettazione e dello sviluppo del prodotto, oltre alla gestione della rete di partner che realizzano progetti di fabbrica intelligente con la piattaforma Opera MES. Permette di elaborare una grande mole di dati rilevati in tempo reale e di identificare prontamente le anomalie che avvengono durante l'intero processo produttivo. Opera è un prodotto MES internazionale, completo ed avanzato per la realizzazione di progetti MES nell'ottica dell'Industria 4.0 e della trasformazione digitale. Come descritto nella figura 3.3, la piattaforma Opera funge da ponte di collegamento verticale tra il livello delle *business applications* (livello 4) e i livelli fisici e di automazione (livelli 1 e 2). A livello orizzontale, mette in comunicazione tutte le aree funzionali della fabbrica per una gestione completa ed integrata: *Production Management*, *Material Management*, *Maintenance Management* e *Quality Management*.

Figura 3.4: Logo di Opera MES

Opera MES utilizza logiche di comunicazione, architetture e protocolli standard per l'integrazione con altri dispositivi e sistemi, aderendo agli standard internazionali industriali. Si tratta di un sistema configurabile, progettato e sviluppato con moderne tecnologie *web-based* in ambito software e con la piattaforma *Microsoft .NET*. Tale sistema è potenziato con l'utilizzo di strumenti di intelligenza artificiale, per trasformare Opera in un sistema MES intelligente, capace di guidare la fabbrica in modo proattivo. Gli operatori possono avere una visione completa del processo produttivo attraverso postazioni web dislocate

Figura 3.5: Screenshot di esempio di Opera MES

nella fabbrica e postazioni web di analisi e supervisione che permettono di accedere ai dati tramite dashboard dinamiche. Attraverso l'interconnessione con i macchinari è possibile acquisire e inviare dati in automatico per consentire un controllo in tempo reale e un'analisi dettagliata delle performance per raggiungere l'eccellenza operativa [48]. La struttura di Opera MES è formata da sei componenti principali:

- **Production:** permette di gestire, tracciare e monitorare tutte le attività di produzione svolte all'interno dell'impianto per consentire una supervisione in tempo reale della produzione.

- **Material:** permette la gestione completa del flusso dei materiali in entrata e in uscita dal magazzino, in modo da accedere a informazioni costantemente aggiornate sulla disponibilità di materiale da utilizzare in produzione.

- **Maintenance:** permette di gestire, tracciare e monitorare tutte le attività di manutenzione per ridurne i costi e i guasti accidentali, in modo da migliorare il livello di utilizzo delle risorse.

- **Quality:** permette di gestire, rilevare e monitorare in modo automatico i parametri di controllo qualità durante il processo produttivo, al fine di garantire la conformità del prodotto.

- **Device Connection:** interconnessione bidirezionale con macchine, attrezzature, linee di produzione e dispositivi in generale, per l'acquisizione automatica di dati e l'invio di istruzioni dal MES ai dispositivi.

- **AI Integration:** utilizzo delle tecnologie emergenti relative all'Industria 4.0 (Internet of Things, Cloud Computing, Big Data Analytics, Intelligenza Artificiale, Machine Learning e Deep Learning) per applicare tecniche predittive sui dati e analisi del processo produttivo, allo scopo di rendere la fabbrica intelligente.

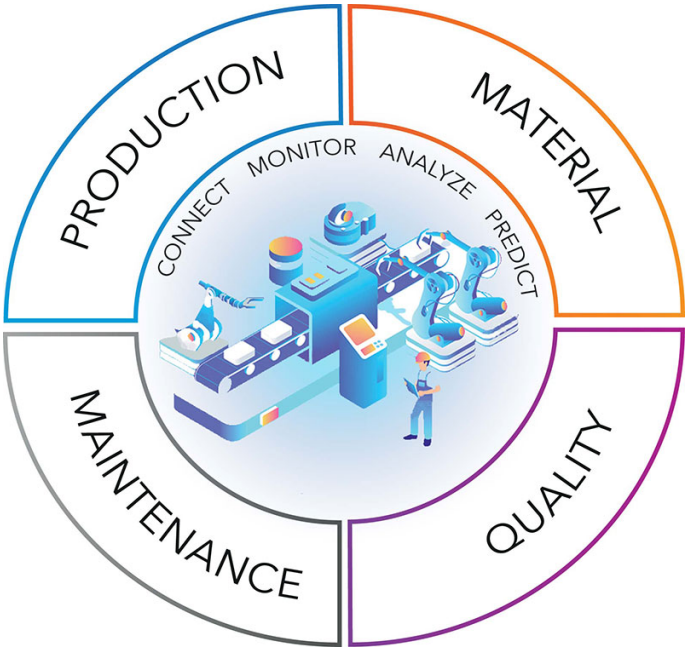

Figura 3.6: Componenti funzionali di Opera MES

3.1.3 Macchinari industriali per lo stampaggio plastico a iniezione

Il sistema di manutenzione predittiva realizzato è stato applicato ed adattato ad una determinata tipologia di macchinari industriali, si tratta di presse a iniezione per lo stampaggio plastico, in particolare una pressa elettrica *BMB eKW* [49]. Tali macchinari, da cui sono stati estratti i dati, si trovano presso l'azienda cliente *STS Tecnopolimeri* di Jesi (Ancona), che utilizza Opera MES. Le sue principali attività riguardano lo stampaggio e la produzione di stampi per iniezione. Strutturalmente è composta da un ufficio tecnico, reparti produttivi e di logistica che permettono di offrire al cliente una soluzione completa, dallo sviluppo alla produzione di un qualsiasi articolo in plastica [50].

Figura 3.7: Logo di STS Tecnopolimeri

Un polimero è una macromolecola costituita da gruppi molecolari che possono essere uguali o diversi e sono uniti a catena con ripetizioni dello stesso tipo di legame covalente. La maggior parte dello stampaggio ad iniezione si concentra sui polimeri termoplastici. Per non perdere le proprietà fisiche specifiche del materiale, esso viene fornito al macchinario sotto forma di *granulato* che può contenere parti di materiale riciclato, materiale vergine ed eventualmente pigmenti per dare colore all'articolo in plastica da produrre. La pressa per lo stampaggio a iniezione è suddivisa in due gruppi principali: l'*unità di iniezione* e l'*unità di chiusura*. La più utilizzata è quella di tipo orizzontale, dove il gruppo di iniezione è posto alla destra dell'operatore, mentre quello di chiusura alla sua sinistra. Il gruppo di iniezione è l'insieme delle componenti che contribuiscono alla fase di *plastificazione* del polimero: vite di plastificazione, tramoggia di alimentazione, motore per l'iniezione, miscelatore ed essiccatore del materiale plastico. Il gruppo di chiusura invece è costituito da un semi stampo fisso collegato all'unità di iniezione, una

parte mobile collegata alla chiusura a ginocchiera, le centraline per il raffreddamento, gli estrattori per l'espulsione del prodotto finito o altre tipologie di manipolatori esterni e il motore per la regolazione del gruppo di chiusura. Il macchinario è gestito da un PLC che regola tutti i parametri durante le varie fasi del processo di produzione.

Figura 3.8: Pressa a iniezione BMB eKW Full Electric

Le principali funzionalità del macchinario sono descritte di seguito:

- **Miscelazione:** vengono mescolate insieme parti di polimero vergine, materiale riciclato, pigmenti colorati ed eventuali altri materiali.
- **Essicazione:** deumidificazione del polimero miscelato per eliminare le particelle umide che possono creare difetti nel prodotto stampato.
- **Caricamento:** tramite la tramoggia di alimentazione il materiale viene convogliato verso la vite per la plastificazione.
- **Regolazioni:** si regola la velocità e la pressione del materiale all'interno della vite, questi controlli sono essenziali per evitare imperfezioni e produzione di scarti.
- **Plastificazione:** la vite per la plastificazione fluidifica il materiale per effetto delle resistenze elettriche e del riscaldamento dovuto all'attrito.

- **Iniezione:** il materiale plastico viene iniettato dentro lo stampo per l'azione di un cilindro idraulico o elettrico.
- **Raffreddamento:** una centralina gestisce il raffreddamento dello stampo.
- **Estrazione:** il gruppo di chiusura esegue le operazioni di apertura dello stampo, estrazione del pezzo e chiusura della parte mobile dello stampo.
- **Unità di governo:** composta da microprocessori e sistemi di controllo numerico.

Le operazioni di miscelazione, essicazione e caricamento sono indispensabili per mantenere elevata la fluidità del materiale e le caratteristiche meccaniche del prodotto stampato. L'elemento principale della pressa a iniezione è la *vite punzonante*, essa è suddivisa in tre zone differenti: alimentazione, compressione e dosaggio. La vite fluidifica il materiale plastico e lo inietta all'interno dello stampo tramite un ugello. Nella camera di iniezione rimane sempre una piccola quantità di polimero, in modo da sfruttare l'*effetto cuscino*, che ha il compito di trasmettere con continuità la pressione della vite al polimero nello stampo. Lo *stampo* è una matrice costituita da due gusci (chiamati semi stampi), che viene modellata per realizzare un pezzo che abbia una determinata forma. Lo stampo ha un costo estremamente elevato, che risulta essere pari quasi al costo dell'intero macchinario. La maggior parte del tempo di ciclo è rappresentata dalla fase di raffreddamento, essenziale per ottenere un prodotto finito privo di difetti. L'estrazione può essere a caduta, manuale o con estrattori che staccano il pezzo dallo stampo e lo posizionano sul nastro trasportatore, ad esempio bracci robotici. Il ciclo di produzione del macchinario può essere riassunto in tre fasi principali:

- **Caricamento e plastificazione:** vengono chiusi lo stampo e l'ugello, applicando una determinata forza di chiusura. Dopo la fase di miscelazione, il granulato passa dalla tramoggia al cilindro, all'interno del quale la vite provvede a plasticizzarlo e a farlo muovere verso la zona di iniezione situata all'altra estremità della stessa. Mano a mano che il fuso entra nella camera di iniezione la vite arretra, finché non si attiva un sensore che controlla il volume da iniettare.
- **Iniezione e mantenimento in pressione:** terminato il processo di plastificazione la vite inverte la corsa e fa avanzare il materiale fuso verso la zona di iniezione. In

questa fase la vite funge da pistone perché muove il materiale mantenendolo ad una pressione corretta. Una *valvola di non ritorno* posizionata in testa alla vite evita che il materiale fuso fluisca all'interno del cilindro di plastificazione. Attraverso l'apertura dell'ugello, il polimero viene iniettato all'interno dello stampo utilizzando una elevata pressione di iniezione e la vite può iniziare a plastificare materiale per il ciclo successivo.

- **Raffreddamento ed estrazione:** viene mantenuta una pressione anche in questa fase del processo per contrastare i ritiri del materiale dovuti al raffreddamento. Dopo che il materiale raggiunge una rigidità sufficiente avviene l'espulsione del prodotto finito.

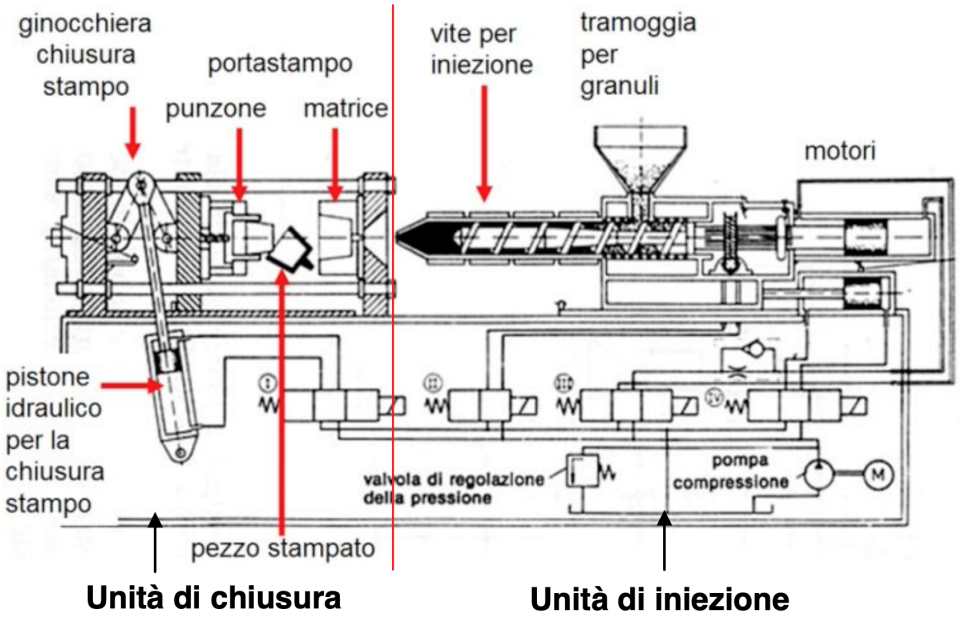

Figura 3.9: Schema di funzionamento di una pressa a iniezione

L'intero ciclo di produzione è estremamente rapido, in modo da realizzare nel minore tempo possibile il maggior numero di pezzi. In ogni zona del macchinario sono presenti numerosi sensori, i cui dati sono stati utilizzati per la realizzazione del sistema di manutenzione predittiva. Esistono diversi tipi di presse ad iniezione, le più moderne sono ad

azionamento elettrico e consentono un controllo del processo più veloce ed accurato, oltre ad un notevole risparmio energetico. Vi sono poi le presse ad azionamento totalmente idraulico e le presse dette *ibride*, con le unità di chiusura e di iniezione che sono in parte ad azionamento elettrico e in parte ad azionamento idraulico. La pressa su cui è stato realizzato il sistema di manutenzione predittiva è di tipo elettrico, mentre le altre presse presenti nell'azienda cliente STS Tecnopolimeri delle quali sono stati estratti i dati sono di tipo ibrido [51].

3.2 Data Understanding

In questa seconda fase del modello CRISP-DM avviene la comprensione dei dati da utilizzare. Dopo aver completato la precedente fase di Business Understanding, ora l'attenzione è rivolta ad identificare, raccogliere e analizzare i set di dati che possono essere utili per raggiungere gli obiettivi del progetto. Questa fase è composta dai seguenti task:

- **Raccolta dei dati iniziali:** si acquisiscono i dati necessari e se necessario si caricano nello strumento di analisi utilizzato.
- **Descrizione dei dati:** si esaminano i dati, documentandone proprietà e caratteristiche. Ad esempio il formato dei dati, il numero di record o il significato di ogni campo.
- **Esportazione dei dati:** si approfondisce la comprensione dei dati, interrogandoli, visualizzandoli e identificando le relazioni presenti nei dataset.
- **Verifica della qualità dei dati:** si controlla se i dati estratti sono puliti o se presentano anomalie, documentando eventuali problemi di qualità.

In questa fase si sono innanzitutto estratti con apposite query SQL i dati dal database Microsoft SQL Server nel quale erano memorizzati, successivamente si è eseguita una catalogazione e documentazione degli stessi per comprenderne meglio il significato. Infine si sono esplorati i dati estratti tramite una loro analisi, utilizzando alcune metriche di statistica descrittiva, grafici a linee e istogrammi di densità, verificandone la qualità.

3.2.1 Esportazione dei dati dal database

Tutti i dati registrati dai sensori del macchinario vengono inviati tramite il PLC e il protocollo di comunicazione *Euromap* ad Opera MES, che si occupa di salvarli nel database Microsoft SQL Server. In tale database di circa 25 GB di dimensione, il sistema MES registra ogni aspetto relativo al funzionamento del macchinario, come gli articoli prodotti e i guasti avvenuti. Dopo aver compreso il funzionamento del database Microsoft SQL Server, la relativa struttura e il significato delle diverse tabelle, si sono progettate e implementate le query SQL per l'estrazione dei dati. Tutte le query sono state eseguite su un *dump* del database, in modo da lavorare direttamente sulle macchine di Open Data. Grazie alle informazioni comprese nelle fasi precedenti, relative all'analisi del dominio del problema, si è scelto di esportare le tipologie di dati che sono descritte di seguito. I dati da esportare sono stati scelti pensando a come dovranno essere aggregati successivamente tra loro, poiché nelle fasi seguenti sarà necessario creare dei *DataFrame* contenenti le letture di ogni sensore durante le produzioni con e senza guasti.

- **Sensori:** sono i dati relativi ai campionamenti registrati dai sensori presenti nel macchinario industriale. Sono stati esportati i dati di tutte le 110 tipologie di sensori disponibili.

- **Difetti:** sono i dati relativi ai difetti degli articoli prodotti, suddivisi in base alla loro tipologia. Sono stati esportati i dati di tutte le 18 tipologie di difetti presenti.

- **Produzioni con guasto:** sono i dati relativi alle produzioni eseguite dal macchinario che hanno presentato dei guasti. Tali guasti sono suddivisi in base alla loro causa e sono stati esportati i dati di tutte le 32 tipologie di guasti registrati.

- **Produzioni corrette:** sono i dati relativi alle produzioni eseguite dal macchinario che non hanno presentato guasti e sono andate a buon fine. Sono stati esportati i dati delle 2 tipologie di produzioni corrette presenti.

I dati esportati sono generati dalle letture dei sensori o prodotti direttamente da Opera MES, quelli relativi ai difetti e alle produzioni con guasto ad esempio sono inseriti direttamente dagli operatori che lavorano a stretto contatto con il macchinario. Le query

SQL sono state implementate sotto forma di script, in modo da automatizzare l'esecuzione delle stesse per le diverse tipologie di dati e i diversi macchinari. Alcuni estratti del codice SQL utilizzato sono riportati nel capitolo dedicato all'implementazione. Tutti i dati descritti sono stati esportati sotto forma di file CSV per i quattro macchinari industriali per lo stampaggio plastico a iniezione presenti presso l'azienda cliente STS Tecnopolimeri (macchinari 0009, 0011, 0063 e 0064). Come descritto precedentemente, il sistema di manutenzione predittiva è stato comunque adattato ad uno solo di questi macchinari (macchinario 0009), ma può funzionare anche per gli altri allenando nuovamente il modello con i rispettivi dati.

3.2.2 Catalogazione dei dati

Tutti i dati esportati nei relativi file CSV sono stati catalogati e documentati. Questo task è stato svolto per comprendere meglio a cosa si riferisse ogni tipologia di dato esportato. I dati relativi al macchinario considerato sono stati registrati da febbraio 2019 a ottobre 2020, mentre i dati degli altri macchinari sono stati registrati da giugno 2018 a ottobre 2020. Complessivamente i dati esportati ammontano a 3 GB. Per quanto riguarda i dati dei sensori, essi si suddividono in *actuals* e *setpoints*. I sensori actuals sono la tipologia di sensori di maggiore interesse per lo scopo del sistema che si vuole realizzare, poiché misurano i segnali rilevati istante dopo istante e variano di conseguenza. Invece i sensori setpoints risultano essere molto più costanti, poiché si riferiscono a parametri che possono essere impostati direttamente dall'operatore che lavora con il macchinario. Per ogni file CSV esportato, relativo ad una tipologia di dato, si è prodotto un file di report in lingua inglese, nel quale sono riportate le informazioni principali su tale dataset:

- **Sensori:** nome del file, nome del sensore, identificativo del macchinario, identificativo dello strumento, identificativo del canale, descrizione del canale, valore minimo, valore massimo, descrizione del sensore e descrizione delle colonne. In questa tipologia di dataset sono presenti due colonne: valore numerico letto dal sensore e istante temporale nel quale è avvenuta la lettura.

- **Difetti:** nome del file, nome del difetto, identificativo della causale, descrizione della causale, descrizione del difetto e descrizione delle colonne. In questa tipologia

di dataset sono presenti quattro colonne: quantità di prodotto scartato, identificativo dell'articolo, istante temporale di inizio della produzione dell'articolo con difetto e istante temporale di fine della produzione dell'articolo con difetto.

- **Produzioni con guasto:** nome del file, nome del guasto, identificativo della causale, descrizione della causale, descrizione del guasto e descrizione delle colonne. In questa tipologia di dataset sono presenti tre colonne: identificativo dell'articolo, istante temporale di inizio della produzione con guasto e istante temporale di fine della produzione con guasto.

- **Produzioni corrette:** nome del file, nome della produzione, descrizione della produzione corretta e descrizione delle colonne. In questa tipologia di dataset sono presenti tre colonne: quantità di prodotto versata, istante temporale di inizio della produzione corretta e istante temporale di fine della produzione corretta.

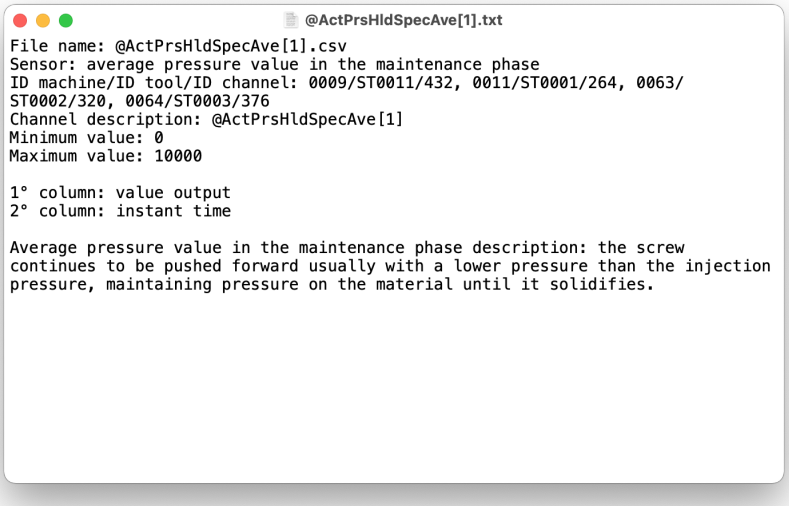

Figura 3.10: Esempio del report di catalogazione dei dati dei sensori

Questa catalogazione e documentazione di tutti i dati esportati dal database è risultata essere utile per avere un'idea più chiara dei dati da utilizzare e per velocizzare il lavoro

nelle fasi successive, consultando la documentazione prodotta per comprendere subito il contenuto dei dataset. Inoltre, tale documentazione può essere utile per progetti ed estensioni future. Di seguito sono riportati alcuni esempi dei dataset esportati:

- **Sensori:** le principali categorie di sensori sono relative a dosatura, riempimento, mantenimento, chiusura dello stampo, apertura dello stampo e sensori di carattere più generale. Tra le 110 tipologie di sensori vi sono ad esempio pressioni in fase di mantenimento, pressioni in fase di plastificazione, quota di dosatura, velocità di riempimento, temperature di zone della vite punzonante, temperature di zone dello stampo, temperatura olio, quota minimo cuscino, quota commutazione, tempo di raffreddamento, tempo di mantenimento, tempo di ciclo e corsa di prerisucchio.

- **Difetti:** tra le 18 tipologie di difetti degli articoli prodotti vi sono ad esempio aloni, striature, bave, bruciature, bolle d'aria, graffi, ritiri, materiale umido, deformazioni, rotture e saldature errate.

- **Produzioni con guasto:** tra le 32 tipologie di guasti ai macchinari vi sono ad esempio interruzioni di acqua, interruzioni di aria, interruzioni elettriche, anomalie nella pressa, guasto elettrico, guasto meccanico, pulizia camera, ugello otturato, pezzo su stampo, anomalie al termoregolatore e guasti non causalizzati.

- **Produzioni corrette:** sono presenti 2 tipologie di produzioni corrette per ogni macchinario industriale. La prima tipologia è relativa alle produzioni corrette classificate come *lavorazione*, che ad esempio potrebbero includere produzioni che hanno realizzato articoli con difetti ma senza guasti al macchinario. La seconda tipologia invece riguarda produzioni che sono completamente corrette e non hanno presentato alcun tipo di problematica.

3.2.3 Analisi dei dati

Dopo la catalogazione e documentazione dei dati, si è creato uno script Python per eseguire alcune analisi sui dati e trarne ulteriori informazioni, anche a livello grafico. In particolare sono stati analizzati tutti i dataset esportati relativi ai sensori. Lo script

realizzato utilizza le librerie Pandas, Seaborn, Matplotlib, e Numpy. Per ogni dataset analizzato crea una nuova directory con i seguenti output:

- **File di report:** un file di testo che include il nome del file a cui si riferisce e le metriche di statistica descrittiva che sono state calcolate. Tali informazioni includono conteggio dei record, media, deviazione standard, valore minimo presente, valore massimo presente e i percentili (25%, 50% e 75%). Un esempio è mostrato nella figura 3.11.
- **Grafico a linee:** un'immagine del grafico a linee che rappresenta l'andamento dei campionamenti di un sensore nel corso del tempo. Sull'asse delle ascisse è indicato il tempo e sull'asse delle ordinate il valore letto dal sensore. Un esempio è mostrato nella figura 3.12.
- **Istogramma di densità:** un'immagine dell'istogramma che rappresenta la densità dei valori letti dal sensore a cui è sovrapposta la funzione normale. Sull'asse delle ascisse è indicato il valore letto dal sensore e sull'asse delle ordinate la relativa densità. Un esempio è mostrato nella figura 3.13.

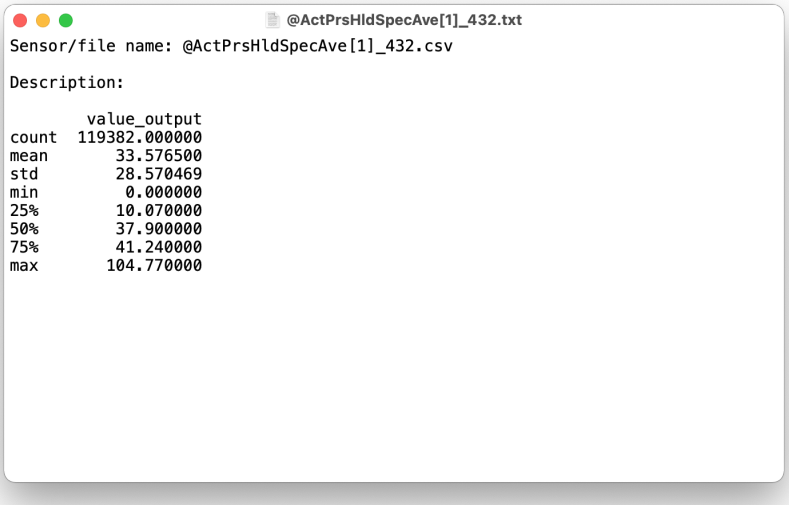

Figura 3.11: Esempio del report di analisi dei dati dei sensori

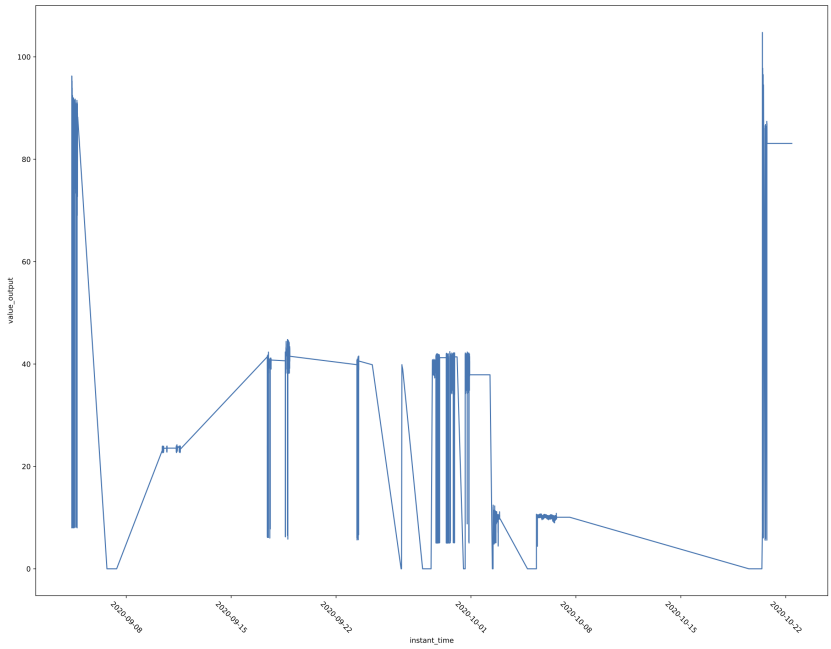

Figura 3.12: Esempio di grafico a linee

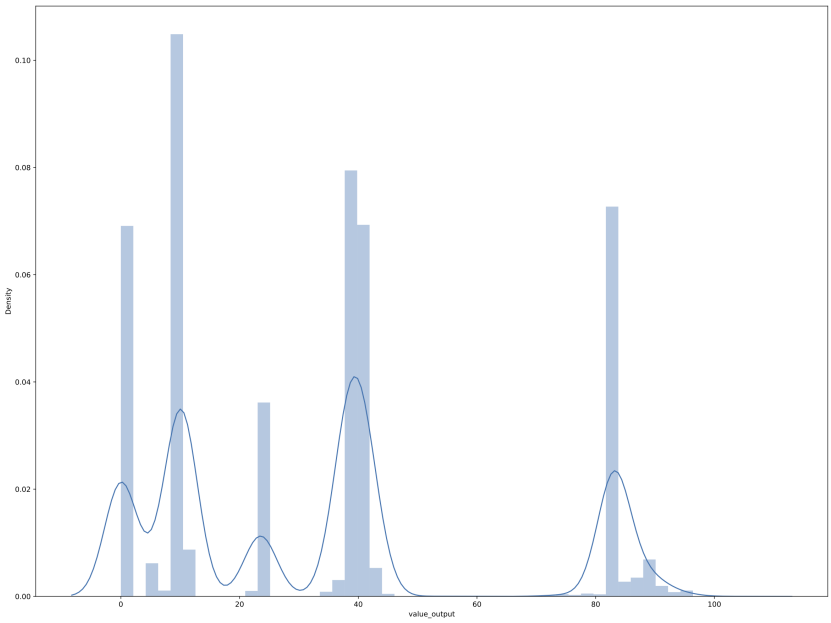

Figura 3.13: Esempio di istogramma di densità

I risultati prodotti sono utili per comprendere le caratteristiche statistiche di ogni dataset, inoltre possono essere utilizzati nella successiva fase di preparazione dei dati per effettuare determinate scelte progettuali.

3.3 Data Preparation

In questa terza fase del modello CRISP-DM avviene la preparazione dei dataset finali che si utilizzeranno durante la successiva fase di Modeling, a partire dai dati iniziali. Questa fase è composta dai seguenti task:

- **Selezione dei dati:** si determinano quali dataset verranno utilizzati, documentando i motivi della scelta di inclusione o esclusione degli stessi.
- **Pulizia dei dati:** si correggono o rimuovono i valori errati.
- **Costruzione di nuovi dati:** si creano nuove caratteristiche che possono essere utili da includere nei dataset finali, ad esempio ricavando la media dei dati.
- **Creazione dei dataset finali:** si creano i dataset finali combinando tra loro i dati da più fonti diverse, secondo opportune logiche.
- **Formattazione dei dati:** si formattano i dati in base alle necessità. Ad esempio, convertendo le stringhe in valori numerici per poter eseguire operazioni matematiche.

Uno dei principali obiettivi di questa fase è stato la riduzione della quantità di dati da utilizzare rispetto a quelli esportati e documentati nelle fasi precedenti, applicando tecniche di correlazione ai dati dei sensori e tecniche di clustering ai dati delle produzioni con guasto, queste operazioni verranno approfondite nei prossimi capitoli. L'altro principale obiettivo di questa fase è stato la costruzione dei DataFrame finali. Innanzitutto si è scelto di utilizzare tutti i dati relativi alle produzioni corrette e di escludere l'utilizzo dei dati relativi ai difetti degli articoli prodotti, in quanto non direttamente rilevanti per raggiungere lo scopo del progetto. Nonostante ciò, come descritto in precedenza, tali dati sono stati comunque estratti e documentati in modo da poter essere utilizzati per eventuali estensioni future come l'implementazione di un sistema di *qualità predittiva*. Si è

scelto quindi di utilizzare i dati sui sensori, sulle produzioni con guasto e sulle produzioni corrette. Grazie ai risultati ottenuti con le analisi eseguite nella fase precedente di Data Understanding, è stato possibile ridurre il numero dei dataset relativi ai sensori considerati. A seguito di tali analisi, controllando sia gli indici statistici che i grafici prodotti precedentemente, si sono esclusi 23 sensori che presentavano caratteristiche non rilevanti o con problematiche. In particolare, tali sensori erano soprattutto di tipo setpoints, i cui valori cambiavano molto poco nel corso del tempo o erano perfettamente costanti. Altri dataset invece erano vuoti, presentavano anomalie strutturali o ancora potevano essere derivabili da altri set di dati, non apportando direttamente nuove informazioni.

3.3.1 Correlazione dei sensori

Per ridurre ulteriormente il numero dei dataset relativi ai sensori del macchinario industriale si è verificata la presenza o meno di correlazione tra coppie di sensori. In questo modo è stato possibile escludere i sensori tra loro correlati, mantenendone soltanto uno. Riducendo la quantità dei dati utilizzati si risparmia spazio di archiviazione, oltre a permettere un training più rapido. Ad esempio, uno dei componenti principali del macchinario industriale, cioè lo stampo, presenta numerosi sensori di temperatura localizzati in zone diverse. Tali sensori, specialmente quelli posizionati uno accanto all'altro risultano tra loro correlati. Questo significa che all'aumentare o al diminuire della temperatura registrata da un sensore, aumenta o diminuisce proporzionalmente anche la temperatura registrata dal sensore vicino. Di conseguenza, non è utile utilizzare entrambi questi dataset ma ne è sufficiente soltanto uno, in grado di rappresentare le informazioni rilevate. La correlazione è il grado in cui due variabili sono tra loro correlate. Nel senso più ampio, la correlazione è qualsiasi relazione statistica, causale o meno, tra due variabili. Il *coefficiente di correlazione* è quindi una misura statistica usata per quantificare la forza della relazione tra due variabili. I valori del coefficiente sono compresi tra -1 e 1. Una correlazione di valore -1 mostra una perfetta correlazione negativa, mentre una correlazione di valore 1 mostra una perfetta correlazione positiva. Infine una correlazione di valore 0 non mostra alcuna relazione lineare tra il movimento delle due variabili. Per verificare la presenza della correlazione si sono utilizzati due coefficienti:

- **Coefficiente di Pearson:** il coefficiente di correlazione di Pearson è una statistica che misura la correlazione lineare tra due variabili x e y. Ha un valore compreso tra +1 e -1. Un valore di +1 indica una correlazione lineare positiva totale, 0 non è una correlazione lineare e -1 è una correlazione lineare negativa totale. La correlazione di Pearson può valutare soltanto una relazione lineare tra due variabili continue. Una relazione è lineare solo quando una variazione in una variabile è associata a una variazione proporzionale nell'altra variabile.

- **Coefficiente di Spearman:** il coefficiente di correlazione di Spearman è una misura statistica non parametrica della correlazione del rango. Misura il grado di relazione tra due variabili e l'unica ipotesi richiesta è che siano ordinabili e, se possibile, continue. Valuta in che misura la relazione tra due variabili possa essere descritta utilizzando una funzione monotona, cioè una relazione che esegue una delle seguenti operazioni: all'aumentare del valore di una variabile, lo stesso fa il valore dell'altra variabile, oppure all'aumentare del valore di una variabile, il valore dell'altra variabile diminuisce, ma non esattamente a un tasso costante. Mentre in una relazione lineare il tasso di aumento o di diminuzione è costante.

Per testare la correlazione tra i sensori si è progettato e realizzato uno script Python che utilizza le librerie Pandas, Seaborn, Matplotlib, Numpy e Scipy. Tale script, eseguito in *multithreading* poiché presenta tempi di calcolo considerevoli, genera tutte le possibili coppie di sensori e modella i due dataset da confrontare. Essi vengono ordinati in modo che gli istanti temporali nei quali sono avvenuti i campionamenti siano il più vicino possibile tra i due dataset, in modo che possano essere confrontati tra loro, durante questo processo i due dataset vengono resi di dimensioni uguali. Infine, per ogni coppia di sensori crea una nuova directory con i seguenti output:

- **File di report:** un file di testo che include i risultati dei coefficienti di correlazione di Pearson e Spearman con la descrizione del livello di correlazione, i relativi tempi di esecuzione e il tempo di esecuzione totale. Un esempio è mostrato nella figura 3.14.

- **Scatterplot:** un'immagine del grafico a punti congiunti da linee che rappresentano l'andamento dei campionamenti dei due sensori della coppia nel corso del tempo,

in modo da poterli confrontare tra loro a livello visivo. Sull'asse delle ascisse è indicato il tempo e sull'asse delle ordinate il valore letto dal sensore. Un esempio è mostrato nella figura 3.15.

- **Boxplot:** due immagini che rappresentano i boxplot dei dataset dei due sensori che si stanno confrontando. Viene usato per descrivere la distribuzione del campione tramite un rettangolo diviso in due parti, dal quale escono due segmenti. Tale rettangolo è delimitato dal primo e dal terzo quartile, diviso al suo interno dalla mediana. Invece i segmenti sono delimitati dal minimo e dal massimo dei valori. Un esempio è mostrato nella figura 3.16.

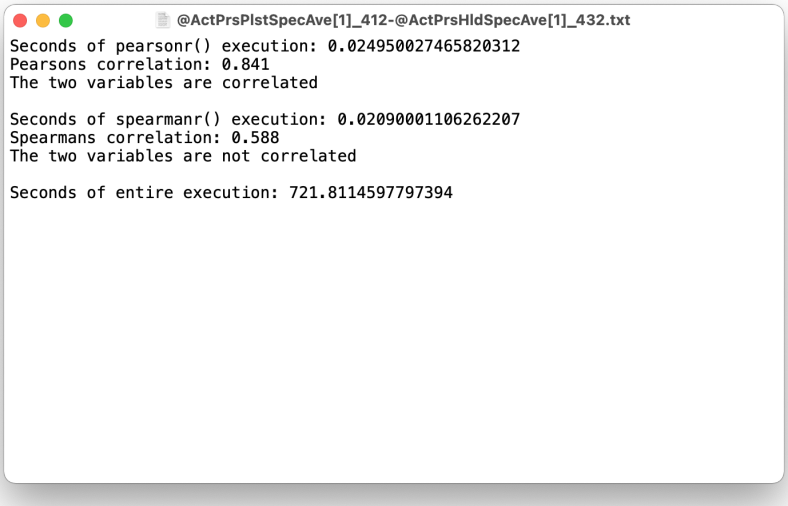

Figura 3.14: Esempio del report di correlazione dei sensori

Nello script Python sono state definite diverse fasce di risultato comprese tra -1 e 1, in modo da identificare meglio il livello di correlazione tra i sensori. Inoltre i risultati globali di tutte le coppie di sensori vengono riportati dallo script su due appositi file di output, nei quali vi sono gli elenchi dei sensori correlati secondo Pearson e secondo Spearman. Dall'analisi dei risultati della correlazione tra i sensori, osservando i grafici prodotti e basandosi sul funzionamento del macchinario industriale, si sono rimossi numerosi sensori

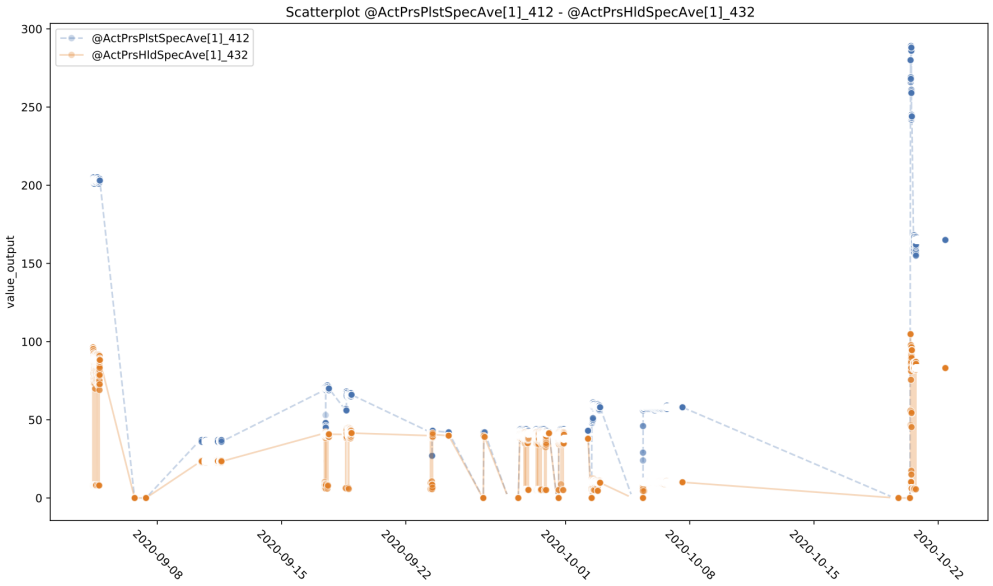

Figura 3.15: Esempio di scatterplot

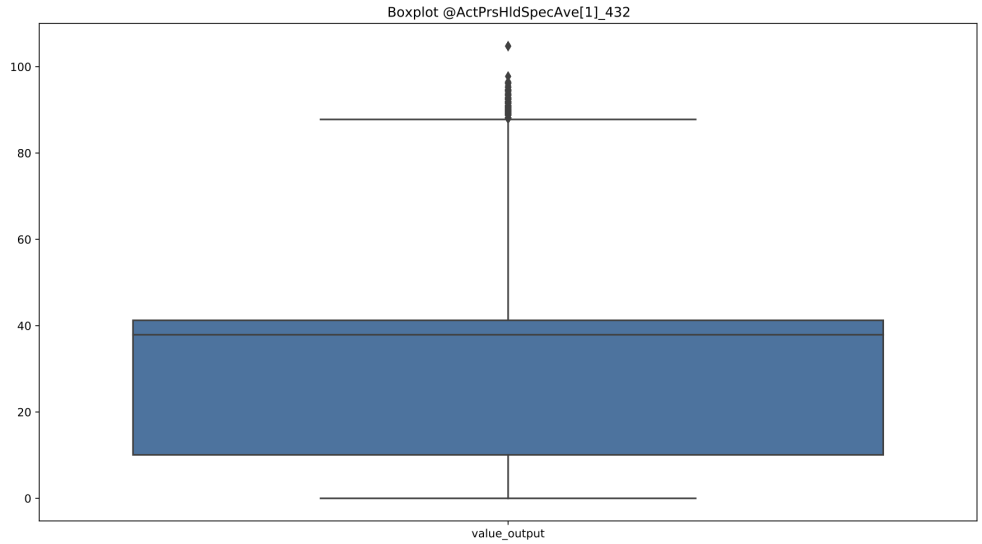

Figura 3.16: Esempio di boxplot

fino a selezionarne 8 che fossero significativi. Per tali ragionamenti ci si è confrontati anche con il referente dell'azienda cliente STS Tecnopolimeri, Dott. Matteo Pigliapoco, il quale ha maggiore esperienza relativamente al funzionamento del macchinario industriale da cui provengono i dati. Tutti gli script realizzati nelle fasi precedenti e successive sono dinamici, di conseguenza il numero e la tipologia dei sensori scelti può essere modificato agevolmente in base alle esigenze. Gli 8 sensori che sono stati selezionati e che verranno utilizzati nelle fasi successive sono descritti di seguito:

- **Quota massima di pressione:** pressione massima alla quale la materia plastica grezza, solitamente sotto forma di granuli, avanza verso lo stampo dall'unità di iniezione.

- **Quota cuscino:** svolge un ruolo importante, i 2-3 mm di fuso ancora presenti nella camera di iniezione permettono il trasferimento della pressione dalla vite al polimero nelle cavità dello stampo.

- **Quota picco di pressione:** pressione massima raggiunta nel macchinario industriale per lo stampaggio plastico a iniezione.

- **Temperatura zona 2:** è la temperatura della seconda zona del cilindro che contiene la vite punzonante. Esso può essere suddiviso in quattro diverse zone, in cui vengono applicate le resistenze elettriche, partendo dalla zona di carico appena sotto la tramoggia fino alla parte adiacente allo stampo.

- **Temperatura zona 4:** è la temperatura della quarta zona del cilindro che contiene la vite punzonante. Esso può essere suddiviso in quattro diverse zone, in cui vengono applicate le resistenze elettriche, partendo dalla zona di carico appena sotto la tramoggia fino alla parte adiacente allo stampo.

- **Temperatura zona stampo 1:** temperatura in un'area specifica dello stampo. Le macchine per lo stampaggio di materie plastiche esaminate hanno 8 diverse zone dello stampo, nelle quali viene campionata la relativa temperatura.

- **Tempo di dosatura:** tempo necessario per completare la fase di dosaggio, cioè il passaggio del fuso nella parte anteriore della vite punzonante.
- **Tempo di riempimento:** tempo necessario per il riempimento dello stampo.

3.3.2 Clustering dei guasti

Come descritto in precedenza, i dati relativi alle produzioni con guasto sono dati *causalizzati*. Questo significa che per ogni guasto dovrebbe essere specificata la relativa causa, in modo da determinare la tipologia di guasto avvenuto. I dati esportati però presentano una serie di problematiche che si è cercato di risolvere, in particolare:

- **Guasti non causalizzati:** la maggior parte dei dati relativi alle produzioni con guasto in realtà non risultano causalizzate. Questo significa che al guasto non è associata una causa specifica e di conseguenza viene segnalato come guasto generico. Tale problematica è causata da errori in fase di raccolta dei dati dal macchinario, poiché gli operatori che lavorano a contatto con esso spesso non segnalano correttamente la tipologia di guasto. Tale situazione potrebbe essere risolta automatizzando questo processo e organizzando la produzione in modo che gli operatori segnalino, almeno inizialmente, i tipi di guasti del macchinario per poi realizzare un modello in grado di compiere automaticamente questa operazione.
- **Fermi macchina:** tra i dataset dei guasti sono presenti molti dati che in realtà non si riferiscono a guasti, ma bensì ai cosiddetti fermi macchina. Si tratta di situazioni in cui il macchinario industriale è attivo ma non sta producendo, non essendo impostato alcun programma da eseguire. Ad esempio momenti in cui l'operatore è in pausa pranzo o vi sono cambi di turno. Tali situazioni sono registrate erroneamente come guasti, per risolvere questa problematica è necessario un cambio alla configurazione del macchinario.

Si è quindi tentato di correggere tali dati non causalizzati ed eseguire una pulizia degli stessi, applicando un filtro in grado di rimuovere i dati relativi ai fermi macchina. Per questo scopo si sono utilizzate tre diverse tecniche di clustering: K-Means, Agglomerative Clustering e DBSCAN. Il funzionamento di tali tecniche è stato descritto nel secondo

capitolo del presente elaborato. Si è progettato e realizzato uno script con Python e la libreria Scikit-learn, che applicasse dinamicamente tali tecniche di clustering al dataset dei guasti non causalizzati, modificandone i parametri all'interno di un range per individuare la combinazione migliore. Lo scopo è stato quello di associare ogni cluster ad una specifica tipologia di guasto. Lo script produce tre differenti directory, una per ogni tecnica di clustering, contenente i seguenti output:

- **Scatterplot dei cluster:** una serie di immagini che rappresentano gli scatterplot prodotti dalla tecnica di clustering applicata. Sull'asse delle ascisse è indicato il tipo di articolo prodotto e sull'asse delle ordinate la durata temporale del guasto. In questo modo si è cercato di identificare un particolare guasto in base alla sua durata e all'articolo prodotto. Inoltre si è utilizzata la tecnica *one-hot encoding*, per mappare le stringhe relative al nome dell'articolo prodotto in valori numerici. Alcuni esempi sono mostrati nelle figure 3.17, 3.18 e 3.19.

- **Grafico a linee del coefficiente di Silhouette:** un'immagine che rappresenta il grafico a linee dell'andamento dell'indice di Silhouette. Sull'asse delle ascisse è indicato per K-Means e Agglomerative Clustering il numero dei cluster, mentre per DBSCAN il numero dei minpoints. Sull'asse delle ordinate invece è indicato il valore del coefficiente di Silhouette. Un esempio è mostrato nella figura 3.20.

Tramite il *coefficiente di Silhouette* si è cercato di valutare il numero appropriato di cluster o di minpoints. Si tratta di una misura della coesione e della separazione dei cluster, quantificando quanto un dato è associato al cluster corretto, valutandone la vicinanza agli altri punti del cluster e la lontananza dai punti di altri cluster. I valori del coefficiente di Silhouette sono compresi tra -1 e +1, risultati più grandi indicano che i campioni sono più vicini ai loro cluster che ad altri cluster. La decisione sul numero di cluster da utilizzare è stata guidata da una combinazione di conoscenza del dominio e metriche di valutazione del cluster. Dai risultati di applicazione delle tecniche di clustering si è riscontrato che i migliori output sono stati per K-Means quelli con un numero di cluster compreso fra 10 e 13. Per Agglomerative Clustering quelli con un numero di cluster compreso fra 13 e 16. Infine, per DBSCAN quelli con un numero di minpoints compreso fra 19 e 20.

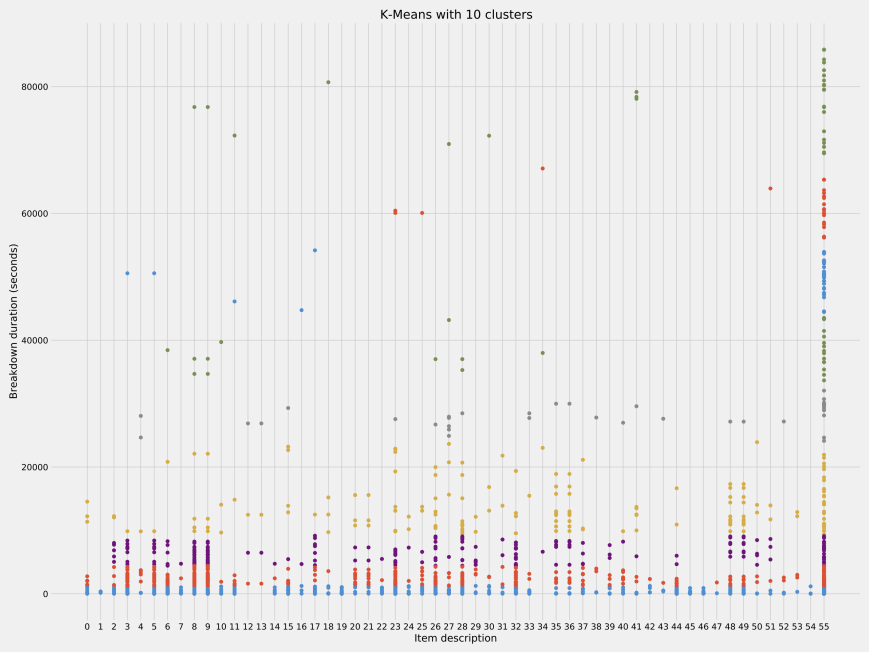

Figura 3.17: Esempio di scatterplot dei cluster con K-Means

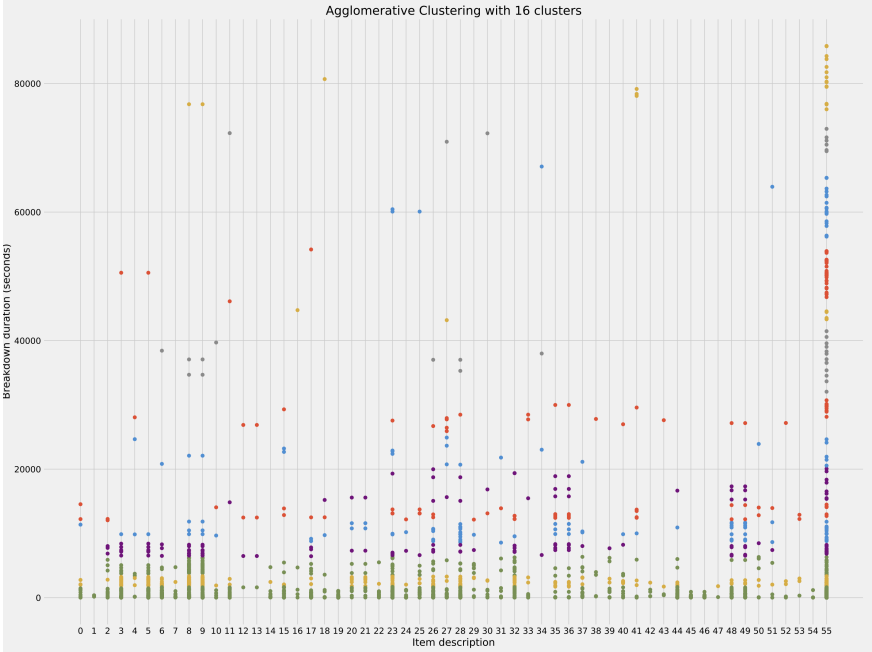

Figura 3.18: Esempio di scatterplot dei cluster con Agglomerative Clustering

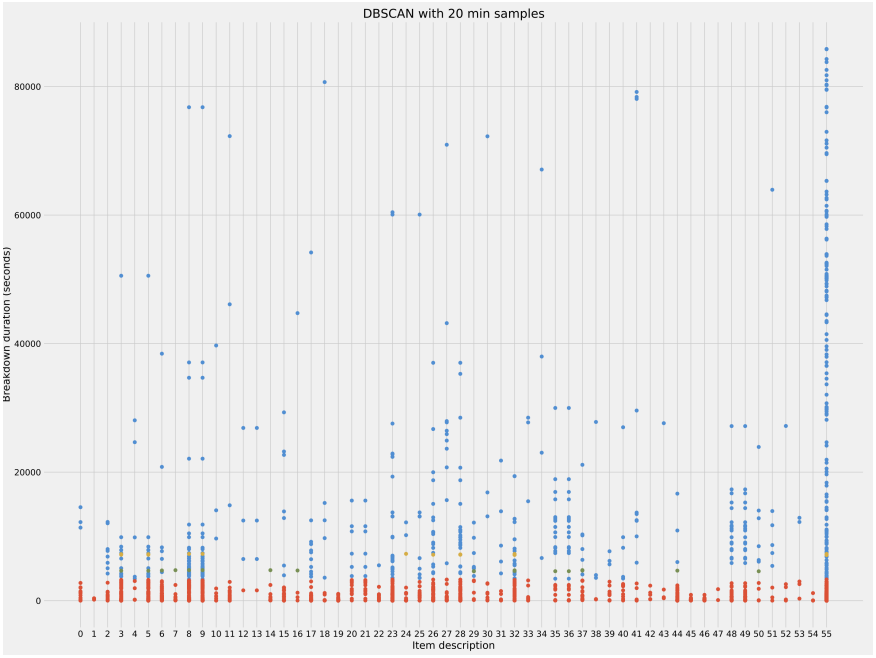

Figura 3.19: Esempio di scatterplot dei cluster con DBSCAN

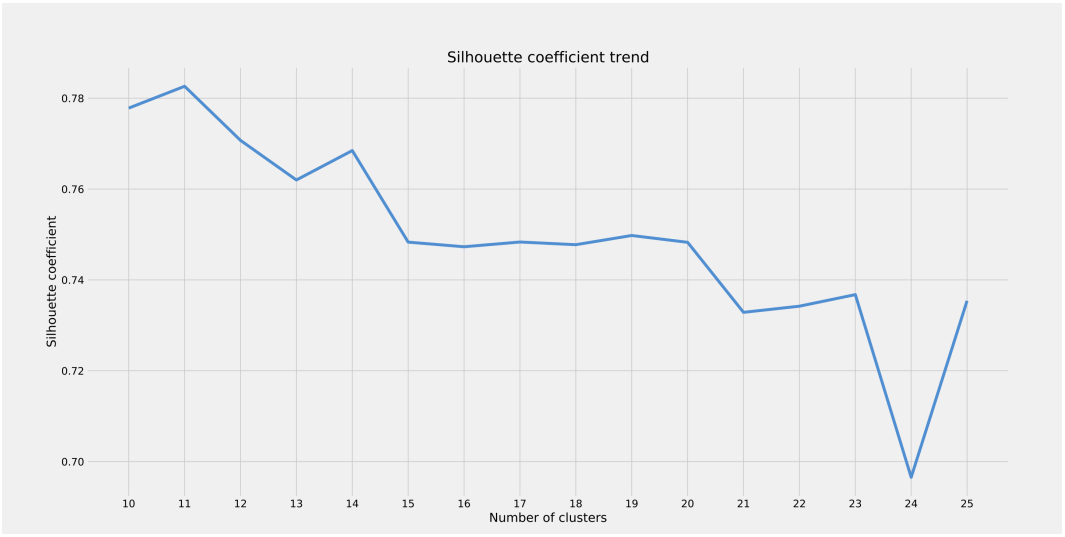

Figura 3.20: Esempio di grafico a linee del coefficiente di Silhouette con K-Means

Quest'ultima tecnica è risultata quella che ha prodotto i risultati peggiori, il parametro eps si è impostato al valore 100. Tale valore è stato determinato calcolando la distanza dagli n punti più vicini per ogni punto, ordinando i risultati. Successivamente si è prodotto il grafico dei risultati, osservando il punto in cui il cambiamento è più pronunciato, cioè il *punto di massima curvatura* e tale valore è stato scelto come eps. Per l'analisi dei risultati prodotti si è avuto un confronto con il referente dell'azienda cliente STS Tecnopolimeri, il quale possiede maggiore esperienza nel contesto dei macchinari industriali utilizzati. A causa della complessità del dominio del problema non è stato possibile determinare con una buona precisione a quali guasti si riferissero determinati cluster. Di conseguenza non è stato possibile individuare una soglia che funzionasse da filtro per i fermi macchina, cioè i finti guasti. Nonostante ciò si è ipotizzato che tali finti guasti si possano individuare nel cluster posizionato più in alto negli scatterplot, il quale ha durate compatibili con possibili fermi macchina. A causa dell'esito di queste analisi, i dati relativi ai fermi macchina rappresentano del rumore nei dataset che rimarrà anche nelle fasi successiva del progetto. Relativamente alla mancata determinazione della causa di ogni guasto, il sistema di manutenzione predittiva mostrerà un avviso di imminente guasto generico. Sarà compito dell'operatore determinare la tipologia di guasto, in base alla sua conoscenza del dominio e interpretando i dati provenienti dai sensori e mostrati graficamente nella dashboard sul client, posizionata in prossimità del macchinario.

3.3.3 Costruzione dei DataFrame

L'ultimo task relativo alla fase di Data Preparation riguarda la creazione dei DataFrame che verranno utilizzati nella successiva fase di Modeling. Un DataFrame è una struttura che contiene dati etichettati organizzati in righe e colonne, si tratta della struttura dati principale utilizzata dalla libreria Pandas. A partire dai dataset selezionati di sensori, produzioni con guasto e produzioni corrette, si sono realizzati cinque diversi DataFrame con complessità e logiche differenti. Gli script Python per creare tali DataFrame sono stati progettati e implementati in base alle esigenze, mano a mano che venivano testati con i modelli creati. Per questo motivo si è passati più volte dalla fase di Data Preparation alla fase di Modeling, come previsto dal modello CRISP-DM. Si tratta di script dinamici, ai quali possono essere dati in input un numero arbitrario di dataset, ad

esempio se si vuole aumentare il numero dei sensori selezionati. In tutti gli script Python realizzati l'utente può inserire in input i *path* alle directory che contengono i file CSV che si vogliono utilizzare. Tali file vengono letti dagli script e il loro contenuto è importato in appositi DataFrame che vengono concatenati, mischiati, modellati e puliti in caso di righe duplicate o produzioni eccessivamente brevi, a seconda del tipo di DataFrame che si vuole costruire. Al termine di ogni script è applicata la standardizzazione dell'intero DataFrame utilizzando apposite funzioni di *preprocessing* di Scikit-learn, esportando il file *scaler.bin* che verrà utilizzato successivamente. Infine ogni DataFrame è esportato in formato CSV. Alcuni estratti del codice implementato per costruire i cinque DataFrame descritti di seguito sono riportati nel capitolo relativo all'implementazione.

3.3.3.1 DataFrame 1 nominal

Il primo DataFrame creato è il più semplice, lo script riceve in input il numero di produzioni con guasto e il numero di produzioni corrette. Per mantenere una proporzione che sia il più vicina alla realtà è necessario impostare circa il 70% delle produzioni come corrette e il restante 30% con guasto, queste proporzioni valgono anche per il secondo e il terzo DataFrame. Ogni riga rappresenta il primo campionamento degli 8 sensori selezionati, che avviene durante una produzione del macchinario. Vengono riportati i valori nominali letti dai sensori. Una produzione è un intervallo di tempo durante il quale è prodotto un certo numero di articoli. Di conseguenza si è progettata e implementata una apposita funzione che cerca il campionamento più vicino al tempo di inizio della produzione. Ogni colonna del DataFrame si riferisce ad un sensore, mentre l'ultima colonna, chiamata *is_breakdown* assume il valore *true* o *false*, rispettivamente se la produzione è classificata con guasto o come corretta. In fase di test del DataFrame sono state aggiunte due righe e due colonne per calcolare rispettivamente la media e la deviazione standard complessiva di ogni colonna e di ogni riga. Inoltre, sempre in fase di test del DataFrame è stata calcolata la differenza di tempo presente tra l'inizio di ogni produzione e il primo campionamento di ogni sensore. Il DataFrame realizzato è quindi composto da 8 colonne, una per ogni sensore, oltre all'ultima colonna is_breakdown.

3.3.3.2 DataFrame 2 historical

Il secondo DataFrame creato è più complesso, utilizza una serie di storici dei quali è calcolata la media e la deviazione standard, anziché utilizzare direttamente i valori nominali come avviene nel primo DataFrame. Lo script riceve in input il numero di produzioni con guasto che si vogliono considerare, successivamente per ogni produzione con guasto si selezionano altre tre produzioni dal dataset delle produzioni corrette, tramite apposite funzioni create. Si controlla se ogni tripla non sia già stata selezionata in precedenza, inoltre, nel caso in cui le produzioni non contengano campionamenti si seleziona una nuova tripla. Tali tre produzioni sono le seguenti:

- **Produzione precedente:** la produzione corretta immediatamente precedente alla produzione con guasto acclarato. Tale produzione selezionata viene classificata come guasto, impostando la colonna is_breakdown a true. In questo modo il modello per la manutenzione predittiva verrà allenato per riconoscere le produzioni immediatamente precedenti al guasto, che presentano valori letti dai sensori già alterati, in modo da avvisare in anticipo dell'imminente guasto.
- **Produzione precedente alla precedente:** la produzione corretta immediatamente precedente a quella descritta nel primo punto. Tale produzione selezionata viene classificata come corretta, impostando la colonna is_breakdown a false.
- **Produzione successiva:** la produzione immediatamente successiva a quella con guasto acclarato. Tale produzione selezionata viene classificata come corretta, impostando la colonna is_breakdown a false, poiché si considera che il guasto sia stato riparato e il macchinario abbia ripreso il suo regolare funzionamento.

Per ognuno degli 8 sensori selezionati, lo script genera 6 differenti colonne del DataFrame, tali colonne rappresentano gli storici breve (ultimi 10 campionamenti), medio (ultimi 30 campionamenti) e lungo (ultimi 60 campionamenti) di media e deviazione standard. Questi storici sono implementati come array di dimensione fissa che si aggiornano continuamente, scalando di una posizione alla volta. Successivamente, per ogni produzione selezionata, si cercano tutti i campionamenti avvenuti tra il tempo di inizio e il tempo di fine della stessa, aggiungendoli alle rispettive colonne del DataFrame nelle giuste posizioni di riga, in modo che si riferiscano alla relativa produzione. Successivamente si

riempiono le celle vuote, se la colonna di un sensore in una produzione risulta completamente vuota si sostituisce con la media globale della colonna stessa. Altrimenti, se la colonna risulta vuota solo in parte, si sostituiscono le celle vuote con la media locale della stessa colonna per quella specifica produzione. Infine si controlla se sono presenti righe duplicate, rimuovendole e mantenendo soltanto la prima, dando precedenza alla riga classificata come produzione corretta. Il DataFrame realizzato è quindi composto da 48 colonne, 6 per ogni sensore, oltre all'ultima colonna is_breakdown. Nella figura 3.21 è mostrata una rappresentazione grafica della struttura del DataFrame.

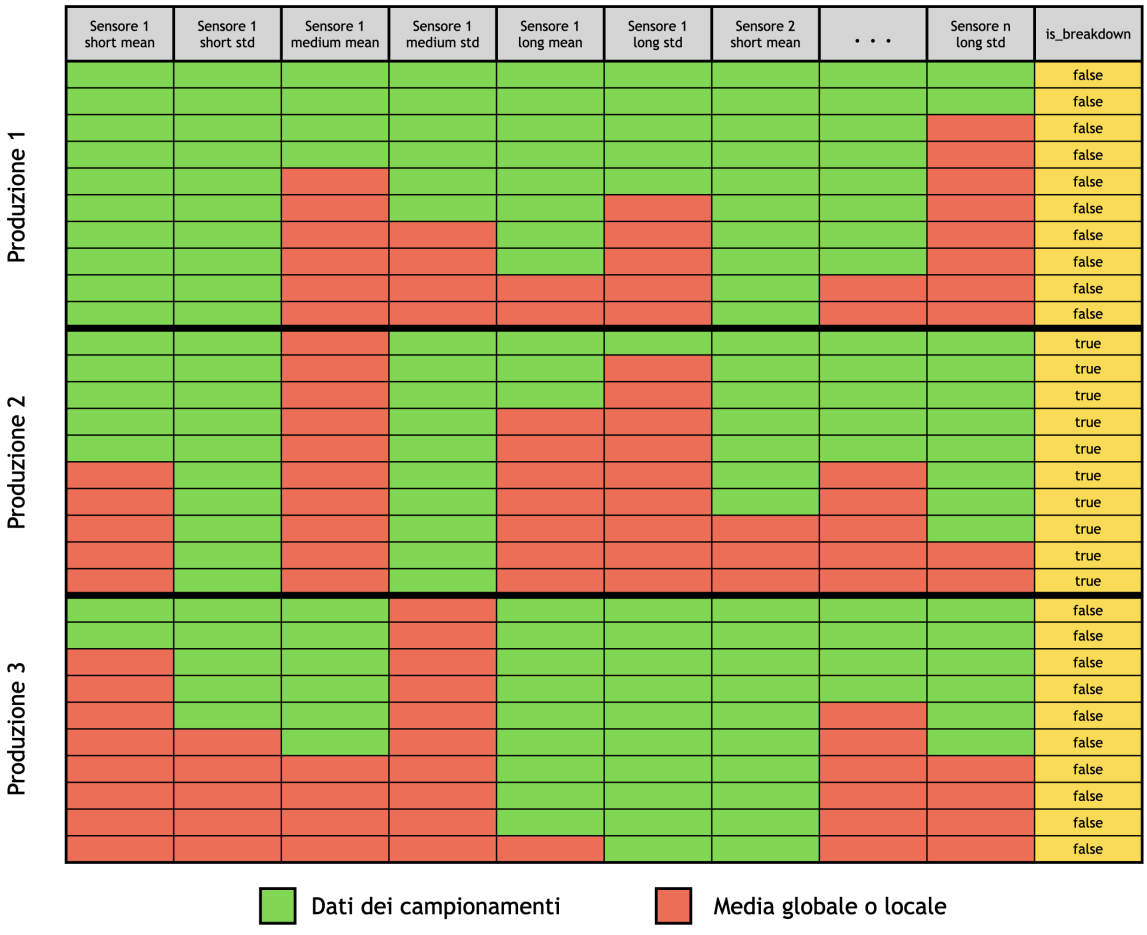

Figura 3.21: Rappresentazione grafica della struttura del DataFrame 2 historical

3.3.3.3 DataFrame 3 nominal sampling

Il terzo DataFrame creato si basa sul funzionamento del secondo DataFrame. L'unica differenza è che è composto dai valori nominali dei campionamenti letti dai sensori in ogni produzione, anziché dalle medie e deviazioni standard dei tre storici. Per questo motivo, anziché 6 colonne per ogni sensore, come avveniva nel secondo DataFrame, in questo caso è presente un'unica colonna per ogni sensore. Inoltre, a differenza del primo DataFrame, nel quale si selezionava solo il primo campionamento di ogni produzione, in questo caso si selezionano tutti i campionamenti che avvengono in una produzione. Il DataFrame realizzato è quindi composto da 8 colonne, una per ogni sensore, oltre all'ultima colonna is_breakdown.

3.3.3.4 DataFrame 4 nominal sampling variant

Il quarto DataFrame creato si basa sul funzionamento del terzo DataFrame, ed è una sua variante. L'unica differenza è che in questo caso si preleva dal dataset delle produzioni corrette solo la produzione successiva a quella con guasto acclarato. Si considera la produzione immediatamente successiva al guasto come certamente corretta, in quanto il guasto è stato riparato. Tale DataFrame verrà utilizzato come input per il training della rete neurale con autoencoder, di conseguenza è un DataFrame che contiene solo produzioni classificate come corrette, quindi con la colonna is_breakdown settata a false. In questo modo la rete neurale artificiale imparerà a ricostruire solo produzioni corrette, individuando le produzioni con guasto quando esse si discostano da determinate soglie di tolleranza. Il DataFrame realizzato è quindi composto da 8 colonne, una per ogni sensore, oltre all'ultima colonna is_breakdown.

3.3.3.5 DataFrame 5 nominal sampling distances

Il quinto DataFrame creato si basa sul funzionamento del quarto DataFrame, ed è una sua variante. Come il DataFrame precedente, si è utilizzato per allenare la rete neurale con autoencoder. Questo nuovo DataFrame è stato realizzato per cercare di migliorare i risultati ottenuti con il precedente DataFrame, forzando ulteriormente le ipotesi fatte. In questo caso lo script Python recupera dal relativo dataset solo le produzioni corrette

che si trovano ad una certa distanza (scelta in input dall'utente) rispetto alla produzione con guasto più vicina, sia in avanti che indietro nel tempo. Si contano quindi quante produzioni corrette sono presenti tra quella che si sta analizzando e il guasto più vicino in avanti e indietro nel tempo. Se tali produzioni sono in numero maggiore o uguale a n, dove n è la distanza scelta in input dall'utente, allora si seleziona la produzione che si sta analizzando, altrimenti la si scarta. Il valore di n utilizzato è stato 5. In questo modo si è cercato di selezionare le produzioni certamente corrette, che fossero influenzate il meno possibile dai guasti, poiché distanti da essi.

3.4 Modeling

In questa quarta fase del modello CRISP-DM avviene la costruzione e il test di vari modelli di apprendimento utilizzando diverse tecniche di Machine Learning e Deep Learning. Questa fase è composta dai seguenti task:

- **Selezione delle tecniche da usare:** si determinano quali algoritmi di Machine Learning e Deep Learning provare, ad esempio Support Vector Machine e reti neurali artificiali.
- **Suddivisione del dataset:** prima di procedere con l'utilizzo delle tecniche di Machine Learning e Deep Learning, è necessario dividere il dataset in training set, test set e validation set.
- **Costruzione del modello:** progettazione e implementazione del modello da allenare e testare, regolandone i parametri. Spesso si utilizzano apposite librerie Python come Scikit-learn e Keras.
- **Valutazione del modello:** solitamente più modelli sono in competizione tra loro, è quindi necessario interpretare i risultati di ogni modello, anche in base alla conoscenza del dominio del problema.

Solitamente, per ottenere un buon modello è necessario iterare il processo più volte, cioè tornare indietro alla fase di Data Preparation per apportare modifiche ai dataset finali e procedere nuovamente con la fase di Modeling per allenare e testare i nuovi dataset.

Durante la realizzazione del progetto si è compiuto tale procedimento. Sono stati utilizzati inizialmente i primi DataFrame costruiti nella fase precedente per provarli durante il Modeling, ripetendo il processo e mano a mano costruendo nuovi DataFrame con lo scopo di migliorare i risultati. Si è scelto di utilizzare alcune tecniche di Machine Learning e Deep Learning. In particolare Support Vector Machine (implementata con la libreria Scikit-learn) e due reti neurali artificiali convoluzionali (implementate con la libreria Keras), in entrambi i casi regolandone i parametri. Gli aspetti teorici e di funzionamento di tali tecniche sono descritti nel secondo capitolo del presente elaborato. I modelli costruiti utilizzando queste tecniche sono stati allenati e testati con i DataFrame realizzati in precedenza. In particolare, i primi tre DataFrame sono stati utilizzati per il training e il testing con SVM e la prima CNN, mentre gli ultimi due DataFrame sono stati utilizzati per il training della seconda CNN con autoencoder. Successivamente, tutti i modelli sono stati testati ulteriormente con nuovi dati e dati mixati esportati dal database, tali dati sono stati registrati dal macchinario industriale nei mesi di novembre e dicembre 2020. I DataFrame creati e utilizzati presentano dimensioni diverse, variando da migliaia a decine di migliaia di record. Alcuni estratti del codice implementato per costruire i tre modelli descritti di seguito sono riportati nel capitolo relativo all'implementazione.

3.4.1 Applicazione di Support Vector Machine

Per applicare e utilizzare Support Vector Machine si è progettato uno script Python utilizzando la libreria Scikit-learn. Innanzitutto lo script, tramite il path inserito dall'utente, importa il DataFrame contenente i dati da utilizzare, in particolare con SVM si sono utilizzati i primi tre DataFrame descritti precedentemente. Successivamente si rimuovono eventuali righe vuote e si etichettano rispettivamente con +1 e -1 i valori *true* e *false* relativi all'esito della produzione associata ad ogni record (colonna is_breakdown). Utilizzando la funzione *train_test_split()* di Scikit-learn si costruiscono il training set e il test set, a cui corrispondono rispettivamente l'80% e il 20% dell'intero DataFrame in input. Per prima cosa si è allenato e testato il modello utilizzando i parametri di default. La funzione kernel utilizzata è stata RBF, descritta nel secondo capitolo del presente elaborato. In seguito si è proceduto regolando i parametri di Support Vector Machine, in particolare C e *gamma*, tramite la tecnica chiamata *Grid Search*. Si tratta di una

tecnica che esegue ripetutamente il training del modello, utilizzando le combinazioni dei parametri C e gamma, i cui valori sono prelevati da appositi insiemi impostati manualmente. In questo modo è possibile trovare la coppia di valori che produce il modello in grado di raggiungere la migliore *accuracy*. Per ogni DataFrame di input con cui è stato allenato il modello di Machine Learning, si sono selezionati ed utilizzati i parametri migliori trovati dalla Grid Search.

C è un parametro di Support Vector Machine richiesto anche quando si utilizzano altri tipi di kernel. Si deve impostare prima della fase di training ed è utilizzato per controllare l'errore di classificazione. Un alto valore di C implica un basso errore ma minore generalizzazione, invece un basso valore di C implica un alto errore ma maggiore generalizzazione. Un basso errore non significa necessariamente un modello migliore, la regolazione del parametro dipende dal dataset che si sta utilizzando. Gamma è un altro parametro di Support Vector Machine richiesto quando si utilizza il kernel RBF. Spesso chiamato anche *sigma*, è utilizzato per dare il peso alla curvatura del confine decisionale. Un basso valore di gamma implica una minore curvatura e quindi una più ampia regione di decisione. Al contrario, un alto valore di gamma implica una maggiore curvatura e quindi una maggiore predisposizione a creare regioni di confini decisionali attorno ai punti. Come per il parametro C, anche la scelta del valore di gamma dipende dal dataset che si sta utilizzando.

Dopo che il modello è stato allenato con i dati del training set, esso viene testato sui dati del test set. Lo script stampa in output i risultati di accuracy, precision, recall e f1-score, raggiunti in fase di testing e il modello allenato viene esportato in un file con estensione *.sav* da utilizzare successivamente. Infine si è realizzato un ulteriore script Python con il solo scopo di testare il modello creato sui nuovi dati esportati dal database e relativi ai mesi di novembre e dicembre 2020. I risultati di tutti i test eseguiti su SVM sono riportati nel capitolo relativo alla validazione.

3.4.2 Applicazione delle 1D Convolutional Neural Networks

Oltre a Support Vector Machine, si è scelto di costruire due modelli di Deep Learning. In particolare sono state utilizzate due reti neurali convoluzionali *1D* che sfruttano approcci differenti. Tali reti sono state scelte confrontandosi con il Prof. Sergei Bezobrazov dell'università bielorussa *Brest State Technical University*, il quale collabora con Open Data quando è necessario effettuare scelte relative al contesto dell'intelligenza artificiale. Entrambe le reti sono state testate con i parametri di default, successivamente regolati per migliorare l'accuratezza della predizione. La struttura e la topologia delle due reti neurali utilizzate sono approfondite nei capitoli successivi, entrambe sono di tipo 1D. Si tratta cioè di reti che risultano essere particolarmente efficaci quando ci si aspetta di ricavare caratteristiche di interesse da segmenti più brevi del dataset complessivo, dove la posizione dell'elemento all'interno del segmento non è di grande rilevanza. Questo approccio si applica bene all'analisi delle sequenze temporali nei dati dei sensori, come è il caso del progetto proposto in questo elaborato. Altri campi di applicazione riguardano l'analisi di qualsiasi tipologia di dati provenienti da segnali e l'*elaborazione del linguaggio naturale*. Le CNN condividono le stesse caratteristiche e seguono lo stesso approccio descritto nel secondo capitolo dell'elaborato, a prescindere dal fatto che siano di tipo 1D, 2D o 3D. La differenza chiave è la dimensionalità dei dati di input e il modo in cui il filtro scorre attraverso i dati. Nel caso delle CNN 1D utilizzate viene specificata l'altezza del filtro (con il parametro *kernel_size*), mentre la larghezza viene adattata automaticamente in modo da coprire tutte le colonne del dataset. In questo modo il filtro scorre sul dataset dall'alto verso il basso. Le due reti scelte, oltre ad utilizzare i layers classici delle CNN descritti nel secondo capitolo, fanno uso di altri tipi di livelli descritti di seguito:

- **Batch normalization:** la normalizzazione batch è un metodo utilizzato per rendere le reti neurali artificiali più veloci e stabili attraverso la normalizzazione del livello di input mediante ricentramento e ridimensionamento. Applica una trasformazione che mantiene la media dell'output vicina al valore 0 e la sua deviazione standard vicina al valore 1.

- **Dropout:** lo scopo di questo livello è assegnare in modo casuale un peso zero ai neuroni della rete. Ad esempio, impostando un tasso di 0,2 il 20% dei neuroni

riceverà un peso zero. Con questa operazione, la rete neurale diventa meno sensibile nel reagire a variazioni minori dei dati, aumentando ulteriormente l'accuratezza sui nuovi dati.

- **Transpose:** il *transpose convolutional layer* è noto anche come *deconvolutional layer*. Tale strato inverte il funzionamento di un livello convoluzionale standard, ovvero se all'output generato da uno strato convoluzionale viene applicata l'operazione di transpose, si ottiene l'input originale. La convoluzione trasposta non inverte la convoluzione standard in base ai valori, ma solo alle dimensioni. Lo strato convoluzionale trasposto è solitamente utilizzato per l'*upsampling*, cioè per generare una feature map di output che ha una dimensione spaziale maggiore di quella della feature map di input.

3.4.2.1 Convolutional Neural Network di classificazione

Per applicare e utilizzare la prima rete neurale convoluzionale di classificazione si è progettato uno script Python utilizzando la libreria Keras. La struttura della rete utilizzata è stata realizzata basandosi sulle risorse pubblicate nel sito web ufficiale della libreria Keras. Innanzitutto lo script, tramite il path inserito dall'utente, importa il DataFrame contenente i dati da utilizzare, in particolare con la prima CNN si sono utilizzati i primi tre DataFrame descritti precedentemente. Successivamente si rimuovono eventuali righe vuote e si etichettano rispettivamente con 1 e 0 i valori *true* e *false* relativi all'esito della produzione associata ad ogni record (colonna is_breakdown). Utilizzando la funzione *train_test_split()* di Scikit-learn si costruiscono il training set e il test set, a cui corrispondono rispettivamente l'80% e il 20% dell'intero DataFrame in input. I DataFrame appena generati si convertono in array Numpy e si applica il *reshape* degli stessi. Successivamente si costruisce il modello definendo la struttura della rete neurale con i vari livelli, tra i quali: *input layer*, *convolutional layer*, *batch normalization layer*, *ReLU layer*, *global average pooling layer* e *fully connected layer*. La funzione di attivazione dell'ultimo layer è di tipo *Softmax*, mentre la funzione di costo utilizzata è la *sparse categorical crossentropy*. La struttura e la composizione dei layers della rete sono mostrate nella figura 3.22.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 48, 1)]           0
_________________________________________________________________
conv1d (Conv1D)              (None, 48, 124)           1364
_________________________________________________________________
batch_normalization (BatchNormalization)  (None, 48, 124)  496
_________________________________________________________________
re_lu (ReLU)                 (None, 48, 124)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 48, 124)           153884
_________________________________________________________________
batch_normalization_1 (BatchNormalization)  (None, 48, 124)  496
_________________________________________________________________
re_lu_1 (ReLU)               (None, 48, 124)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 48, 124)           153884
_________________________________________________________________
batch_normalization_2 (BatchNormalization)  (None, 48, 124)  496
_________________________________________________________________
re_lu_2 (ReLU)               (None, 48, 124)           0
_________________________________________________________________
global_average_pooling1d (GlobalAveragePooling1D) (None, 124)  0
_________________________________________________________________
dense (Dense)                (None, 2)                 250
=================================================================
Total params: 310,870
Trainable params: 310,126
Non-trainable params: 744
_________________________________________________________________
```

Figura 3.22: Struttura della Convolutional Neural Network di classificazione

In seguito si esegue il training del modello, inizialmente lo si è allenato e testato utilizzando i parametri di default, per poi regolarli in modo ottimale. I principali parametri sono stati settati come segue:

- **Epochs:** 100
- **Batch size:** 30
- **Filters:** 124
- **Kernel size:** 10
- **Strides:** 1

- **Padding:** same

Dopo che il modello è stato allenato con i dati del training set, esso viene testato sui dati del test set. Lo script stampa in output i risultati di accuracy, precision, recall e f1-score raggiunti in fase di testing. Il modello allenato viene esportato in un file con estensione *.h5* da utilizzare in seguito e viene prodotto in output il grafico che mostra l'evoluzione dell'apprendimento, riportato nella figura 3.23. Infine si è realizzato un ulteriore script Python con il solo scopo di testare il modello creato sui nuovi dati esportati dal database e relativi ai mesi di novembre e dicembre 2020. I risultati di tutti i test eseguiti sulla prima CNN sono riportati nel capitolo relativo alla validazione [52] [53].

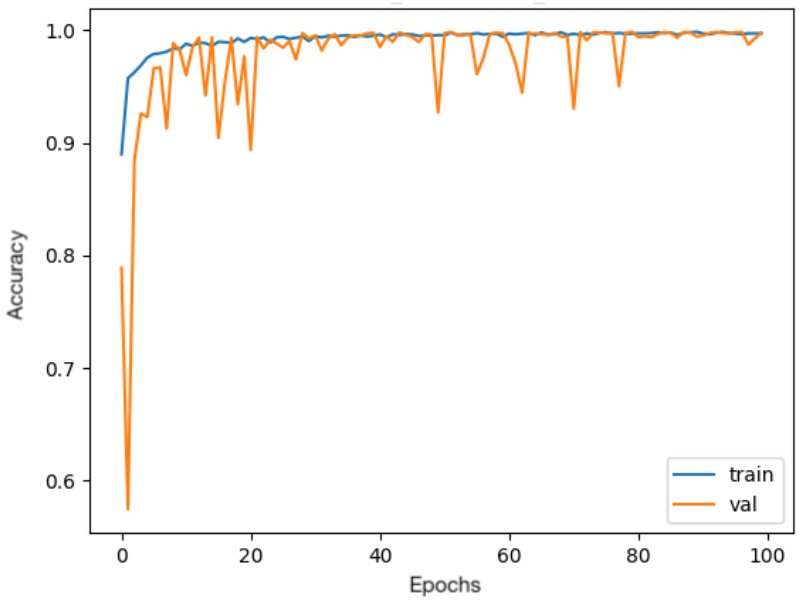

Figura 3.23: Apprendimento della Convolutional Neural Network di classificazione

3.4.2.2 Convolutional Neural Network con autoencoder

Per applicare e utilizzare la seconda rete neurale convoluzionale con autoencoder si è progettato uno script Python utilizzando la libreria Keras. La struttura della rete

utilizzata è stata realizzata basandosi sulle risorse pubblicate nel sito web ufficiale della libreria Keras. Innanzitutto lo script, tramite i path inseriti dall'utente, importa i DataFrame contenenti i dati da utilizzare, in particolare con la seconda CNN si sono utilizzati come training set gli ultimi due DataFrame descritti precedentemente, mentre come test set si è utilizzato il terzo DataFrame. In questo modo i DataFrame di training contengono solo produzioni corrette che la rete neurale con autoencoder deve imparare a riconoscere, mentre il DataFrame di testing contiene sia produzioni con guasto che produzioni corrette. Lo script, tramite un'apposita funzione, modella il training set per creare una serie di sequenze combinando 100 valori di dati contigui nel dataset di addestramento. Successivamente si costruisce il modello definendo la struttura della rete neurale con i vari livelli, tra i quali: *input layer*, *convolutional layer*, *dropout layer* e *transpose convolutional layer*. La funzione di costo utilizzata è la *MSE* (acronimo di *Mean Squared Error*). La struttura e la composizione dei layers della rete sono mostrate nella figura 3.24.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 50, 64)            3648
_________________________________________________________________
dropout (Dropout)            (None, 50, 64)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 25, 32)            14368
_________________________________________________________________
conv1d_transpose (Conv1DTranspose)  (None, 50, 32)     7200
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 32)            0
_________________________________________________________________
conv1d_transpose_1 (Conv1DTranspose)  (None, 100, 64)  14400
_________________________________________________________________
conv1d_transpose_2 (Conv1DTranspose)  (None, 100, 8)   3592
=================================================================
Total params: 43,208
Trainable params: 43,208
Non-trainable params: 0
_________________________________________________________________
```

Figura 3.24: Struttura della Convolutional Neural Network con autoencoder

Il modello costruito è un *convolutional reconstruction autoencoder*, che prende un input di dimensione 100 righe e 8 colonne e ritorna un output con le stesse dimensioni. In seguito si esegue il training del modello, inizialmente lo si è allenato e testato utilizzando i parametri di default, per poi regolarli in modo ottimale. In questa CNN, al contrario della precedente, si utilizza il DataFrame di training sia come input che come *target* poiché si tratta di un modello di ricostruzione. Anche in questo caso le funzioni di attivazione sono di tipo ReLU, i principali parametri sono stati settati come segue:

- **Epochs:** 10
- **Batch size:** 30
- **Filters:** 64, 32, 32, 64
- **Kernel size:** 7
- **Strides:** 2
- **Padding:** same

Dopo che il modello è stato allenato con i dati del training set, viene prodotto in output il grafico che mostra l'evoluzione dell'apprendimento, riportato nella figura 3.25. Successivamente è possibile rilevare le anomalie determinando quanto bene il modello è in grado di ricostruire i dati in input:

- Si trova la perdita MAE (acronimo di Mean Absolute Error) sui campioni di allenamento.
- Si trova il valore massimo di perdita MAE, per ogni colonna, quindi per ogni sensore. Questo rappresenta il caso peggiore relativamente a come si è comportato il modello cercando di ricostruire un campione. Tali valori rappresenteranno le 8 soglie per il rilevamento delle anomalie.
- Se la perdita di ricostruzione per un campione è maggiore di un certo numero di soglie impostate, si può dedurre che il modello sta trattando un campione con cui non ha familiarità. Di conseguenza si classificherà tale campione come un guasto.

Successivamente si prepara il test set per testare il modello allenato. Si creano nuovamente le sequenze come avvenuto per il training set e si ricalcolano i valori di perdita MAE sul test set che è stato ricostruito dal modello. In seguito si selezionano gli indici di tutti i record che sono stati classificati come anomalie. Questo avviene se n sensori di un record, con n scelto dall'utente, superano le relative soglie di errore. Infine lo script controlla la correttezza delle predizioni eseguite confrontandole con il test set originale e conta il numero di *true positives* e *false positives*. A differenza della prima CNN non è stato realizzato un ulteriore script per testare il modello allenato, poiché tutte le operazioni avvengono nello stesso script. Anche in questo caso il modello creato è stato ulteriormente testato sui nuovi dati esportati dal database e relativi ai mesi di novembre e dicembre 2020. I risultati di tutti i test eseguiti sulla seconda CNN sono riportati nel capitolo relativo alla validazione [54]. Le ultime due fasi del modello CRISP-DM, ovvero Evaluation e Deployment, sono descritte nel capitolo relativo alla validazione.

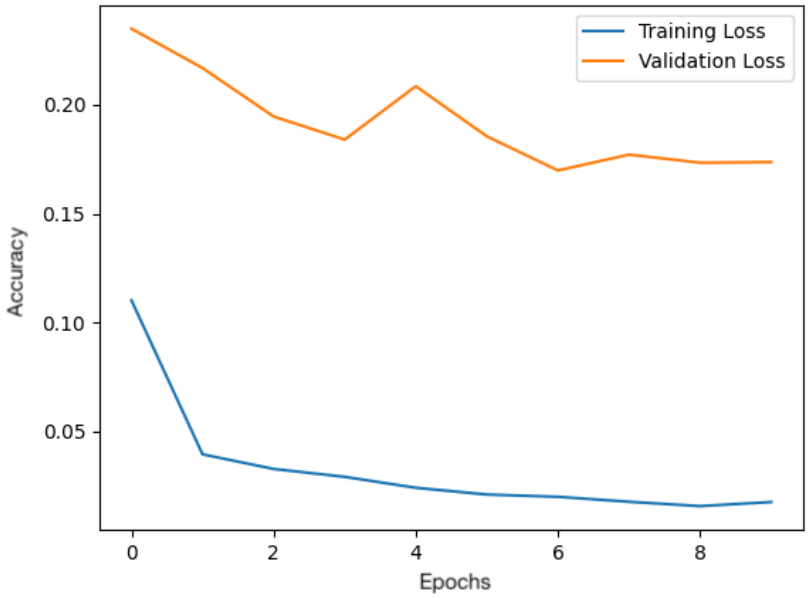

Figura 3.25: Apprendimento della Convolutional Neural Network con autoencoder

Capitolo 4

Implementazione

Nel capitolo precedente si è descritta la fase di progettazione del lavoro svolto, per definire i task eseguiti in ogni fase del modello CRISP-DM, a cui seguiranno nei prossimi capitoli le ultime due fasi, relative al processo di validazione. L'obiettivo di questo capitolo è invece quello di approfondire la fase di implementazione del progetto, quindi gli aspetti più tecnici relativi alle tecnologie utilizzate, al codice scritto e ai diversi linguaggi di programmazione e librerie software di cui si è fatto uso.

4.1 Tecnologie utilizzate

Per lo svolgimento del progetto descritto è stato necessario utilizzare diverse tecnologie e linguaggi di programmazione, che insieme hanno consentito l'implementazione del sistema di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione. Si sono combinate tra loro e utilizzate contemporaneamente più tecnologie, ad esempio per eseguire le query al database, realizzare gli script Python per costruire i DataFrame e allenare i modelli di Machine Learning, o ancora per eseguire il sistema sul cloud computing, implementare il web server e realizzare alcuni mockup. L'intero progetto è stato realizzato e testato sui sistemi operativi Windows 10 e macOS Big Sur, mentre come strumento per il controllo delle versioni si è utilizzato GitHub. Da menzionare infine alcuni software e servizi che sono stati utilizzati poiché il lavoro di tirocinio, come descritto precedentemente, è stato svolto in modalità smart working a causa della

pandemia in corso di COVID-19. Tra tali strumenti rientrano GlobalProtect, utilizzato per collegarsi alla rete VPN aziendale e Microsoft Remote Desktop, per accedere alle macchine situate presso Open Data. Inoltre, sono stati utilizzati alcuni strumenti di comunicazione con i colleghi come Slack e Skype.

4.1.1 Microsoft SQL Server

Microsoft SQL Server è un DBMS (acronimo di *Database Management System*) relazionale, progettato e sviluppato da Microsoft a partire dal 1989. Inizialmente utilizzato per database di medie e di piccole dimensioni, a partire dalla versione 2000 può essere utilizzato anche per la gestione di database di grandi dimensioni. Microsoft SQL Server utilizza una variante del linguaggio SQL standard, denominata T-SQL (acronimo di *Transact-SQL*) e supporta molteplici tipologie di dati. Per realizzare il progetto di manutenzione predittiva si è interrogato questo tipo di database per il recupero dei dati da utilizzare, essendo utilizzato da Opera MES per memorizzare tutti i dati registrati dal sistema. Inoltre si è utilizzato Microsoft SQL Server Management Studio, cioè il software con interfaccia grafica per la configurazione, la gestione e l'amministrazione di tutti i componenti relativi a Microsoft SQL Server. Lo strumento include un editor di script e strumenti grafici che funzionano con oggetti e funzionalità del server. Una delle principali funzionalità di Microsoft SQL Server Management Studio è l'*Object Explorer*, che consente di esplorare, selezionare e agire su qualsiasi oggetto all'interno del server. Può essere utilizzato per osservare visivamente e analizzare la struttura delle query, oltre ad ottimizzare le prestazioni del database. Microsoft SQL Server Management Studio può essere utilizzato per creare nuovi database, alterare gli schemi dei database esistenti aggiungendo o modificando tabelle e indici o ancora per analizzarne le prestazioni. Infine, include alcune funzionalità per scrivere ed eseguire le query SQL tramite una apposita GUI [55].

Figura 4.1: Loghi di Microsoft SQL Server e Python

4.1.2 Python

Python è un linguaggio di programmazione di alto livello orientato agli oggetti. Si tratta di un linguaggio particolarmente adatto per sviluppare script, applicazioni distribuite, e computazione numerica. Python è stato ideato dall'informatico olandese Guido Van Rossum e la prima versione è stata rilasciata nel 1991. Il nome scelto per il linguaggio è dovuto alla passione del suo ideatore per il gruppo comico inglese *Monty Python* e per la loro serie televisiva *Monty Python's Flying Circus*. Si tratta di un linguaggio di programmazione dinamico, semplice e flessibile, inoltre supporta il paradigma orientato agli oggetti, la programmazione strutturata e molte caratteristiche della programmazione funzionale. Il codice sorgente di Python non viene convertito direttamente in linguaggio macchina, ma prima vi è una fase di pre-compilazione in *bytecode*, quest'ultimo viene riutilizzato dopo la prima esecuzione del programma per evitare di reinterpretare ogni volta il sorgente, migliorando così le prestazioni. Una delle principali caratteristiche di Python sono le variabili non tipizzate e l'uso dell'indentazione per la sintassi anziché utilizzare le parentesi, come avviene solitamente negli altri linguaggi di programmazione. Altre caratteristiche particolari di Python sono l'*overloading* di operatori e funzioni, un elevato numero di funzioni di base e librerie disponibili con cui ampliare notevolmente le capacità del linguaggio, oltre a sintassi avanzate come ad esempio *list comprehension* e *slicing*. In Python vi è un forte controllo dei tipi che viene eseguito in *runtime*, inoltre è presente un sistema di *garbage collector* per liberare e recuperare automaticamente la memoria. Spesso Python è definito come un linguaggio di scripting, ma l'elevata quantità di librerie disponibili e la semplicità con cui il linguaggio permette di scrivere software modulare favoriscono lo sviluppo di applicazioni che possono essere anche molto

complesse. Python possiede anche un modulo grafico chiamato *Python Turtle Graphics*. Durante la realizzazione del progetto di manutenzione predittiva, Python si è utilizzato per implementare tutti gli script per la gestione e l'elaborazione dei dati, utilizzando apposite librerie che sono descritte nei prossimi capitoli. In particolare è stata utilizzata la versione 3 di Python. Gli IDE (acronimo di *Integrated Development Environment*) utilizzati per la scrittura e il testing del codice Python sono stati Microsoft Visual Studio Code, Notepad++ e Sublime Text, oltre all'utilizzo diretto del terminale per l'esecuzione degli script [56] [57].

4.1.3 Numpy

NumPy è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata dal data scientist americano Travis Oliphant e la prima versione è stata rilasciata nel 1995. Tale libreria software aggiunge al linguaggio Python il supporto ad array multidimensionali e a grandi matrici, oltre ad una estesa collezione di funzioni matematiche di alto livello che permettono di gestire e manipolare in modo efficiente tali strutture dati. NumPy velocizza i tempi di calcolo fornendo funzioni multidimensionali che operano in modo efficiente sugli array, ottimizzando principalmente i loop interni. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Numpy si è utilizzata insieme al linguaggio Python per implementare gli script relativi alla gestione e all'elaborazione dei dati [58].

Figura 4.2: Loghi di Numpy e Scipy

4.1.4 Scipy

SciPy è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata da Travis Oliphant, Eric Jones, e Pearu Peterson. La prima versione di SciPy è stata rilasciata nel 2001. Si tratta di una libreria software composta da un insieme di algoritmi e strumenti matematici dedicati al linguaggio di programmazione Python. In particolare include moduli dedicati all'ottimizzazione, all'integrazione, all'algebra lineare, funzioni speciali, funzioni per l'elaborazione di segnali e di immagini, risolutori e altri strumenti utilizzati nell'ambito della scienza e dell'ingegneria. La struttura dati di base utilizzata da SciPy è un array multidimensionale fornito dalla libreria NumPy. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Numpy si è utilizzata insieme al linguaggio Python per implementare gli script relativi alla gestione e all'elaborazione dei dati. Ad esempio, durante l'analisi della correlazione per implementare i coefficienti di Pearson e Spearman [59].

4.1.5 Matplotlib

Matplotlib è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata dal neurobiologo americano John Hunter e la prima versione è stata rilasciata nel 2003. Si tratta di una libreria software di *plotting* che utilizza anche la libreria matematica NumPy per la creazione di grafici di tipologie diverse. Ad esempio grafici a linee, istogrammi e scatter plot. Matplotlib rende disponibili funzioni dedicate e orientate agli oggetti per l'inserimento di grafici all'interno delle applicazioni Python, utilizzando toolkit GUI generici. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Matplotlib si è utilizzata insieme al linguaggio Python per implementare gli script relativi all'analisi, alla correlazione e al clustering dei dati, in particolare per la creazione di alcuni grafici [60].

4.1.6 Seaborn

Seaborn è una libreria open source realizzata per il linguaggio di programmazione Python. Si tratta di una libreria software di visualizzazione dei dati basata su Matplotlib, fornisce un'interfaccia di alto livello per disegnare grafici statistici e informativi di

qualità grafica superiore, definendone stili e colori. Inoltre ha la particolarità di integrarsi direttamente con le strutture dati utilizzate dalla libreria Pandas. Le funzioni di plottaggio di Seaborn operano su DataFrame e array contenenti interi dataset ed eseguono internamente la mappatura semantica e l'aggregazione statistica necessarie per produrre grafici informativi. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Seaborn si è utilizzata insieme al linguaggio Python per implementare gli script relativi all'analisi e alla correlazione dei dati, in particolare per la creazione di alcuni grafici [61].

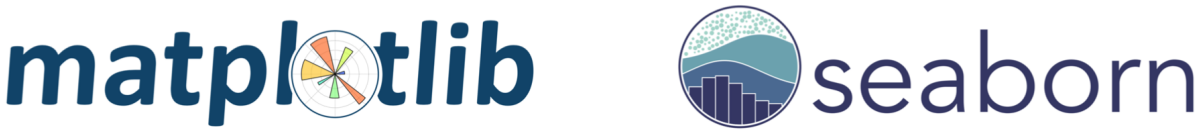

Figura 4.3: Loghi di Matplotlib e Seaborn

4.1.7 Pandas

Pandas è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata dall'informatico americano Wes McKinney e la prima versione è stata rilasciata nel 2008. Il suo nome deriva da *panel data*, termine econometrico per dataset che include osservazioni su più periodi di tempo per gli stessi individui. Si tratta di una libreria software specializzata nella gestione e nell'analisi dei dati. In particolare, offre operazioni e strutture dati per la manipolazione di tabelle numeriche e serie temporali. La libreria è altamente ottimizzata per le prestazioni e i suoi due componenti principali sono *Series* e *DataFrame*. Una Series è essenzialmente una colonna, mentre un DataFrame è una tabella multidimensionale composta da una raccolta di Series. Pandas è utilizzato principalmente per l'analisi dei dati e permette di importare i dataset da vari formati di file come CSV, JSON, SQL e Microsoft Excel. Consente inoltre numerose operazioni di manipolazione dei dati come merging, reshaping e selezione, oltre alla

pulizia del dataset. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Numpy si è utilizzata insieme al linguaggio Python per implementare tutti gli script relativi alla gestione e all'elaborazione dei dati [62].

Figura 4.4: Loghi di Pandas e JupyterLab

4.1.8 JupyterLab

JupyterLab è un ambiente di sviluppo interattivo basato sul web per notebook, codice, dati e progetti Jupyter, la cui prima versione è stata rilasciata nel 2018. JupyterLab è flessibile, permette di configurare e organizzare l'interfaccia utente per supportare un'ampia gamma di flussi di lavoro nei contesti relativi alla Data Science, al calcolo scientifico e al Machine Learning. JupyterLab può essere esteso ed è estremamente modulare, infatti sono disponibili *plugin* che aggiungono nuovi componenti e si integrano con quelli esistenti. In JupyterLab è possibile creare e condividere i Notebook, cioè documenti che contengono codice eseguibile, equazioni matematiche, grafici e testo descrittivo. I Notebook possono essere utilizzati per testare il codice, eseguire la pulizia e la trasformazione dei dati, effettuare simulazioni numeriche, modellazione statistica, visualizzazione dei dati, Machine Learning e molte altre applicazioni. Durante la realizzazione del progetto di manutenzione predittiva, JupyterLab si è utilizzato per creare diversi Notebook, con i quali è stato possibile testare il codice Python e le librerie software utilizzate, prima di realizzare gli script finali da eseguire [63].

4.1.9 Scikit-learn

Scikit-learn è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata dal data scientist francese David Cournapeau e la prima versione è stata rilasciata nel 2007. Si tratta di una libreria software dedicata all'apprendimento automatico, include dozzine di algoritmi e modelli di classificazione, regressione, clustering, random forest e macchine a vettori di supporto. Ogni *estimator* dispone del metodo *fit()*, utilizzato per apprendere i parametri dai dati di addestramento. Inoltre sono presenti i metodi *predict()* e *trasform()* per eseguire le previsioni sui nuovi campioni. Scikit-learn è diventata una delle librerie più utilizzate nell'ambito dell'apprendimento automatico, sia supervisionato che non supervisionato, grazie alla vasta gamma di strumenti che offre, ma anche grazie alla sua API ben documentata, facile da usare e versatile. Risulta essere particolarmente utilizzata per la ricerca accademica poiché è possibile utilizzare il tool per sperimentare molteplici algoritmi modificando soltanto alcune righe di codice. Scikit-learn è progettato per operare insieme alle librerie NumPy e SciPy, inoltre si integra bene con molte altre librerie Python, come Matplotlib e Pandas. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Scikit-learn si è utilizzata insieme al linguaggio Python per implementare gli algoritmi di Machine Learning utilizzati nei relativi script, in particolare Support Vector Machine, K-Means, Agglomerative Clustering e DBSCAN [64].

Figura 4.5: Loghi di Scikit-learn e Keras

4.1.10 Keras

Keras è una libreria open source realizzata per il linguaggio di programmazione Python. È stata ideata dall'ingegnere francese François Chollet e la prima versione è stata rilasciata nel 2015. Si tratta di una libreria software dedicata al Machine Learning e al Deep Learning. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, supporta come back-end le librerie *TensorFlow*, *Microsoft Cognitive Toolkit* e *Theano*. Keras è una libreria pensata per permettere una rapida prototipazione di reti neurali profonde, si concentra sulla facilità d'uso, la modularità e l'estensibilità. Offre infatti una serie di moduli che permettono di sviluppare reti neurali artificiali indipendentemente dal back-end utilizzato, con un linguaggio comune e intuitivo. Keras contiene numerose implementazioni di elementi costitutivi delle reti neurali come livelli, funzioni di attivazione, ottimizzatori e una serie di strumenti per semplificare il lavoro con dati di immagini e testo. Oltre alle reti neurali standard, Keras supporta le reti neurali convoluzionali, ricorrenti e molti altri layers come dropout, batch normalization e pooling. Keras consente la costruzione di modelli di apprendimento eseguibili sui dispositivi mobili iOS e Android, sul web e in ambiente Java. Inoltre permette di utilizzare l'allenamento distribuito di modelli su cluster di unità di elaborazione grafica e unità di elaborazione tensoriale. Durante la realizzazione del progetto di manutenzione predittiva, la libreria Keras si è utilizzata insieme al linguaggio Python per implementare gli algoritmi di Deep Learning utilizzati nei relativi script, in particolare le due reti neurali convoluzionali di classificazione e con autoencoder [65].

4.1.11 Flask

Flask è un framework web open source, ideato dall'informatico austriaco Armin Ronacher, la cui prima versione è stata rilasciata nel 2010. Flask è scritto con il linguaggio di programmazione Python ed è basato sullo strumento *Werkzeug WSGI* e sul motore di template *Jinja2*. È considerato un micro-framework, poiché possiede un nucleo semplice ma estendibile. Non sono presenti strati di astrazione per la base di dati, la validazione dei formulari o qualsiasi altra componente per fornire funzionalità comuni per le quali esistono già librerie di terze parti. Flask supporta comunque estensioni che pos-

sono aggiungere funzionalità all'applicazione sviluppata, come se fossero implementate direttamente dallo stesso Flask. Durante la realizzazione del progetto di manutenzione predittiva, il framework Flask si è utilizzato per implementare il web server nel quale viene eseguito il modello di apprendimento automatico [66].

Figura 4.6: Loghi di Flask e Bootstrap

4.1.12 HTML

HTML, acronimo di *HyperText Markup Language*, è il linguaggio standard di markup utilizzato per creare pagine web. Nato nel 1993, è di pubblico dominio e deriva dall'SGML, un metalinguaggio per definire linguaggi di markup. L'HTML permette di creare documenti ipertestuali, definendone struttura, aspetti grafici, testi, immagini e link attraverso l'inserimento di specifici tag che vengono interpretati dal browser, il quale genera il DOM (acronimo di *Document Object Model*). La sintassi è stabilita dal W3C, che nel corso degli anni ha cercato di definire uno standard comune, insieme alle maggiori aziende e organizzazioni informatiche riunite nel WHATWG. Ogni documento HTML ha una struttura che definisce l'*header*, contenente informazioni di controllo, e il *body*, all'interno del quale è scritto il contenuto vero e proprio della pagina web. Attualmente l'ultima versione pubblicata è HTML 5, che integra funzionalità prima utilizzabili solo tramite estensioni del browser, oltre ad essere pensata per il corretto funzionamento delle pagine sui dispositivi mobili. Durante la realizzazione del progetto di manutenzione predittiva, il linguaggio HTML si è utilizzato per definire il layout della pagina web di configurazione del sistema, testandone la compatibilità con i web browser Google Chrome, Apple Safari e Microsoft Edge [67].

4.1.13 CSS

CSS, acronimo di *Cascading Style Sheets*, è un linguaggio utilizzato per descrivere lo stile di pagine web realizzate in HTML o altri linguaggi di markup. Nato nel 1996, le specifiche del linguaggio sono definite dal W3C, con il CSS è possibile separare il contenuto dei documenti HTML dalla loro presentazione. In questo modo l'HTML è di più facile lettura e risulta più semplice riutilizzare e mantenere il codice dedicato alla presentazione, che può essere inserito in un foglio di stile esterno richiamabile dal documento HTML. È anche possibile scrivere codice CSS direttamente all'interno del file HTML o nei singoli elementi presenti nello stesso. Quando si applicano nuove proprietà di stile agli elementi HTML, questi ultimi sono richiamati tramite vari tipi di selettori, ad esempio di *classe* o *id*. La versione attuale del linguaggio è CSS 3. Durante la realizzazione del progetto di manutenzione predittiva, il linguaggio CSS è stato utilizzato per definire lo stile degli elementi visualizzati sulla pagina web di configurazione del sistema [68].

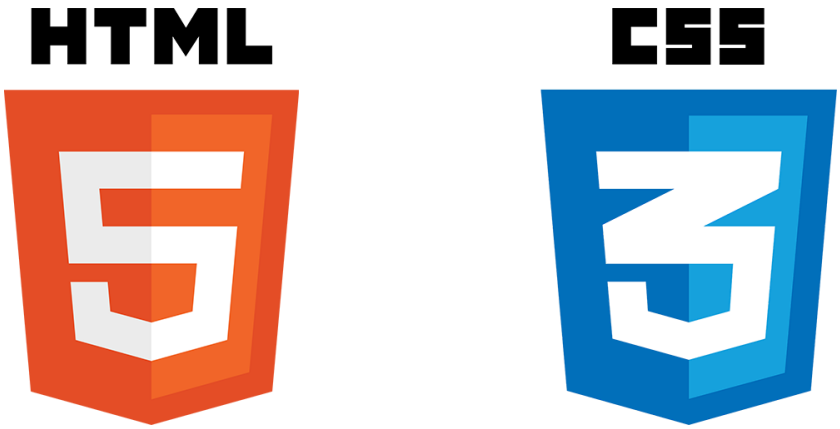

Figura 4.7: Loghi di HTML e CSS

4.1.14 Bootstrap

Bootstrap è un framework open source per la creazione di siti e applicazioni web di vario tipo. Nato inizialmente come progetto interno a Twitter, con lo scopo di uniformare l'interfaccia grafica del social network, nel 2011 è stato rilasciato con licenza open source, per permettere ad altri sviluppatori di contribuire al progetto. Bootstrap è pensato per un utilizzo front-end e contiene strumenti e modelli basati su HTML e CSS come form, bottoni, tabelle, menu e tanti altri elementi tipografici e dedicati all'interfaccia grafica, comodamente implementabili all'interno delle pagine web. Sono presenti alcune estensioni opzionali di JavaScript e inoltre il framework supporta il *responsive design*, in modo che il *layout* del sito web si adatti automaticamente se visualizzato su computer, tablet o smartphone. Bootstrap è compatibile con tutti i moderni browser, utilizza un sistema a griglia per posizionare gli elementi nella pagina web e per utilizzarlo è sufficiente includerlo nel codice HTML. Durante la realizzazione del progetto di manutenzione predittiva, il framework Bootstrap è stato utilizzato per impostare il layout e lo stile dei componenti dell'interfaccia grafica relativa alla pagina web di configurazione del sistema [69].

4.1.15 Postman

Postman è un software disponibile per Windows, macOS e Linux, oltre che utilizzabile direttamente da web browser. Si tratta di uno strumento che permette di testare le API sviluppate in modo semplice e veloce specificandone l'URL. Supporta diversi formati per la comunicazione dei dati come JSON, XML, HTML e offre numerosi metodi, ad esempio GET, POST, PUT e DELETE. Inoltre è possibile impostare e gestire *headers*, *request body* e *response* nella stessa finestra. Tra i vantaggi che offre Postman vi è la cronologia di tutte le chiamate effettuate con le relative autenticazioni utilizzate. Durante la realizzazione del progetto di manutenzione predittiva, il software Postman è stato utilizzato per testare l'esecuzione del sistema in locale e sul cloud computing, inviando le chiamate HTTP POST con i valori letti dai sensori al web server e ricevendo come risposta il risultato della predizione eseguita dal modello di Machine Learning [70].

Figura 4.8: Loghi di Heroku e AWS

4.1.16 Heroku

Heroku è una *Platform as a Service* (PaaS) basata sul cloud computing, fondata nel 2007 da James Lindenbaum, Adam Wiggins e Orion Henry, facente parte del gruppo *Salesforce* dal 2010. Supporta diversi linguaggi di programmazione come Java, Python, PHP, Node.js, Ruby, Go, Scala e Clojure. Si tratta di una delle prime piattaforme che consentono la creazione, l'esecuzione e il ridimensionamento delle applicazioni sviluppate, direttamente sul cloud. Le applicazioni eseguite su Heroku solitamente dispongono di un dominio univoco, utilizzato per instradare le richieste HTTP al contenitore dell'applicazione corretto, chiamato *dyno*. Ogni dyno è distribuito in una struttura composta da diversi server, le cui risorse e specifiche tecniche possono essere configurate in base al piano scelto. Infine, il server Git di Heroku gestisce tutte le push eseguite verso le *repository* delle applicazioni da parte degli utenti autorizzati. Tutti i servizi Heroku sono ospitati sulla piattaforma di cloud computing *EC2* di *Amazon Web Services (AWS)*. Si tratta di una importante sussidiaria di Amazon, che fornisce API e piattaforme di cloud computing su richiesta a privati, aziende e governi tramite il modello *pay-as-you-go*. La tecnologia AWS è implementata in apposite server farm presenti in tutto il mondo. In particolare, il servizio utilizzato da Heroku e chiamato *Amazon Elastic Compute Cloud (EC2)*, consente agli utenti di accedere ad un cluster virtuale di computer, sempre disponibile tramite una connessione ad internet. Le macchine virtuali di AWS emulano la maggior parte delle caratteristiche tecniche di un computer reale, come CPU, GPU, RAM, SSD, sistema operativo, software applicativo precaricato e sono particolarmente adatte ad eseguire applicazioni di Machine Learning e Deep Learning, grazie alla loro potenza computazionale scalabile. Durante la realizzazione del progetto di manutenzio-

ne predittiva, la piattaforma Heroku si è utilizzata per eseguire il sistema costruito sul cloud computing, rendendolo così accessibile via web. L'upload del progetto su Heroku si è eseguito utilizzando i comandi del software Git dal terminale [71].

4.1.17 Balsamiq

Balsamiq è un software disponibile per Windows e macOS, oltre che utilizzabile direttamente da web browser. La prima versione è stata rilasciata nel 2008 e si tratta di un'applicazione che permette la creazione di *wireframe* e *mockup* relativi ad applicazioni desktop, applicazioni mobile e siti web. Consente al progettista di disporre i widget predefiniti all'interno della finestra, utilizzando un editor tramite *drag and drop*. È uno strumento di progettazione rapida dell'interfaccia utente con un basso grado di fedeltà, che riproduce l'esperienza di disegnare su un blocco note o una lavagna, ma utilizzando un computer. Permette di concentrarsi sulla struttura e sul contenuto della pagina, evitando discussioni su dettagli grafici che devono essere definiti successivamente nel processo di sviluppo. Durante la realizzazione del progetto di manutenzione predittiva, il software Balsamiq si è utilizzato per realizzare i tre mockup relativi alla dashboard visualizzata sul client che utilizza Opera MES [72].

Figura 4.9: Loghi di Postman e Balsamiq

4.2 Codice implementato e file

Per implementare il sistema di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione si sono realizzati diversi *script*, scritti prevalentemente con il linguaggio Python, con i quali si è fatto uso di tutte le tecnologie descritte in precedenza. Il funzionamento di tali script è stato descritto nei capitoli relativi a progettazione e validazione. Nei capitoli successivi si riportano gli *snippets* di codice più rilevanti, suddivisi in base al task e al file a cui appartengono.

4.2.1 script_export.sql

Il file contiene tutte le query e gli script SQL utilizzati per l'esportazione dei dati dal database Microsoft SQL Server e per i test eseguiti sui dati contenuti nello stesso. Di seguito si riporta lo script SQL per l'esportazione dei dati relativi a tutte le 110 tipologie di sensori del macchinario 0009.

```
1  -- To allow advanced options to be changed.
2  EXEC sp_configure 'show advanced options', 1;
3  GO
4  -- To update the currently configured value for advanced options.
5  RECONFIGURE;
6  GO
7  -- To enable the feature.
8  EXEC sp_configure 'xp_cmdshell', 1;
9  GO
10 -- To update the currently configured value for this feature.
11 RECONFIGURE;
12 GO
13
14 DROP TABLE tempTable
15 DECLARE @name VARCHAR(255)
16 DECLARE @id INT
17 DECLARE @getid CURSOR
18 SET @getid = CURSOR FOR
19 SELECT cn_codice
20   FROM Canali
21   WHERE ma_codice='0009'
```

```
22 OPEN @getid
23 FETCH NEXT
24 FROM @getid INTO @id
25 WHILE @@FETCH_STATUS = 0
26 BEGIN
27        SELECT IDENTITY(INT, 1,1) AS temp_id, pa.ev_value AS value_output, pa.
              ev_t_event AS instant_time
28        INTO tempTable
29        FROM MntrProcessEvent AS pa
30        WHERE pa.ma_codice = 0009 AND pa.ev_line = @id
31        ORDER BY pa.ev_t_event
32
33        SELECT @name=REPLACE(cn_descanale, ' ', '')
34        FROM Canali
35        WHERE cn_codice=@id;
36
37        DECLARE @insertCommand1 VARCHAR(255) = 'bcp "SELECT value_output,
              instant_time FROM OperaSTS.dbo.tempTable ORDER BY temp_id" queryout C:\
              Datasets\' + @name + '_';
38        DECLARE @insertCommand2 VARCHAR(255) = Concat(@insertCommand1,@id);
39        DECLARE @insertCommand3 VARCHAR(255) = '' + @insertCommand2 + '.csv -c -T -
              t; -S127.0.0.1,52019';
40        EXEC master.dbo.xp_cmdshell @insertCommand3;
41        DROP TABLE tempTable
42    FETCH NEXT
43    FROM @getid INTO @id
44 END
45 CLOSE @getid
46 DEALLOCATE @getid
```

Di seguito si riporta lo script SQL per l'esportazione dei dati relativi a tutte le 32 tipologie di produzioni con guasto del macchinario 0009.

```
1 DROP TABLE tempTable
2 DECLARE @name VARCHAR(255)
3 DECLARE @id VARCHAR(255)
4 DECLARE @getid CURSOR
5 SET @getid = CURSOR FOR
6 SELECT ca_codice
```

```
7   FROM Causali
8   WHERE ca_tipo = 'FM'
9  OPEN @getid
10 FETCH NEXT
11 FROM @getid INTO @id
12 WHILE @@FETCH_STATUS = 0
13 BEGIN
14        SELECT IDENTITY(INT, 1,1) AS temp_id, ol.ar_codice AS item_id, pr.dp_datain
               AS start_time,
15        pr.dp_datafi AS end_time
16        INTO tempTable
17        FROM Produzione AS pr
18        INNER JOIN Causali AS ca (NOLOCK) ON pr.ca_codice = ca.ca_codice
19        LEFT JOIN Fasi AS fa (NOLOCK) ON fa.fa_id = pr.fa_id
20        LEFT JOIN Pianificati AS pn (NOLOCK) ON pn.pi_id = fa.fa_id
21        LEFT JOIN Lavori AS la (NOLOCK) ON la.la_id = pn.la_id
22        LEFT JOIN OrdiniDiLavoro AS ol (NOLOCK) ON ol.ol_codice = la.ol_codice
23        WHERE pr.ma_codice = 0009 AND ca.ca_tipo = 'FM' AND ca.ca_codice = @id
24        ORDER BY pr.dp_datain
25
26        SELECT @name = REPLACE(REPLACE(REPLACE(REPLACE(ca_descau, '>', ''), '\%', '
              '), '/', '-'), ' ', '')
27        FROM Causali
28        WHERE ca_codice = @id;
29
30        DECLARE @insertCommand1 VARCHAR(255) = 'bcp "SELECT item_id, start_time,
              end_time FROM OperaSTS.dbo.tempTable ORDER BY temp_id" queryout C:\
              Datasets\' + @name + '_';
31        DECLARE @insertCommand2 VARCHAR(255) = Concat(@insertCommand1,@id);
32        DECLARE @insertCommand3 VARCHAR(255) = '' + @insertCommand2 + '.csv -c -T -
              t; -S127.0.0.1,52019';
33        EXEC master.dbo.xp_cmdshell @insertCommand3;
34        DROP TABLE tempTable
35    FETCH NEXT
36    FROM @getid INTO @id
37 END
38 CLOSE @getid
39 DEALLOCATE @getid
```

Di seguito si riportano le query SQL per l'esportazione dei dati relativi alle 2 tipologie di produzioni corrette del macchinario 0009. Gli altri script per l'esportazione dei dati relativi a sensori, difetti, produzioni con guasto e produzioni corrette di tutti i quattro macchinari sono stati implementati in modo simile. Lo stesso vale anche per le esportazioni dei dati nuovi, registrati a partire dal mese di novembre 2020.

```
SELECT pr.dp_qtavrs AS poured_quantity, pr.dp_datain AS start_time, pr.dp_datafi AS
    end_time
FROM Produzione AS pr
WHERE pr.ma_codice = 0009 AND pr.dp_azione = 02
ORDER BY pr.dp_datain

SELECT pr.dp_qtavrs AS poured_quantity, pr.dp_datain AS start_time, pr.dp_datafi AS
    end_time
FROM Produzione AS pr
WHERE pr.ma_codice = 0009 AND pr.ca_codice IS NULL
ORDER BY pr.dp_datain
```

4.2.2 script_analysis.py

Il file contiene lo script Python utilizzato per l'analisi dei dati relativi ai sensori di tipologia actuals del macchinario. Le analisi dei dati relativi ai sensori di tipologia setpoints sono state implementate in modo simile. Di seguito si riportano gli snippets di codice più rilevanti dello script.

```
position_path = os.path.dirname(os.path.realpath(__file__))
Path(os.path.join(position_path, 'Analysis')).mkdir(parents = True, exist_ok = True
    )
Path(os.path.join(position_path, 'Analysis', 'Analysis Actuals')).mkdir(parents =
    True, exist_ok = True)
Path(os.path.join(position_path, 'Analysis', 'Analysis Setpoints')).mkdir(parents =
    True, exist_ok = True)
try:
    path_actuals = input("Enter the path to the actuals folder (e.g. 'path\\folder
        '): ")
except ValueError:
    raise error
```

```
try:
    path_setpoints = input("Enter the path to the setpoints folder (e.g. 'path\\
        folder'): ")
except ValueError:
    raise error
# Actuals analysis
for file_name in glob.glob(os.path.join(path_actuals, '*.csv')):
    Path(os.path.join(position_path, 'Analysis', 'Analysis Actuals', os.path.
        basename(file_name)[:-4])).mkdir(parents = True, exist_ok = True)
    dataset = pd.read_csv(file_name, ';')
    dataset.columns = ['value_output', 'instant_time']
    # Report file
    text_file = open(os.path.join(position_path, 'Analysis', 'Analysis Actuals', os
        .path.basename(file_name)[:-4], os.path.basename(file_name)[:-4] + '.txt'),
         'w')
    text_file.write('Sensor/file name: \%s' \% os.path.basename(file_name))
    text_file.write('\n\nDescription:\n\n' + dataset.describe().to_string())
    text_file.close()
    # Lines chart
    dataset['instant_time'] = dataset['instant_time'].astype("datetime64")
    fig = plt.figure(figsize = (20, 15))
    plt.tight_layout()
    sns.lineplot(data = dataset, x = "instant_time", y = "value_output")
    plt.xticks(rotation = -45)
    plt.savefig(os.path.join(position_path, 'Analysis', 'Analysis Actuals', os.path
        .basename(file_name)[:-4], os.path.basename(file_name)[:-4] + '_lines.pdf')
        , dpi = 300)
    plt.clf()
    plt.close()
    # Histogram of density
    fig = plt.figure(figsize = (20, 15))
    plt.tight_layout()
    sns.distplot(dataset['value_output']);
    plt.savefig(os.path.join(position_path, 'Analysis', 'Analysis Actuals', os.path
        .basename(file_name)[:-4], os.path.basename(file_name)[:-4] + '_normal.pdf'
        ), dpi = 300)
    plt.clf()
    dataset = dataset[0:0]
```

4.2.3 script_correlation.py

Il file contiene lo script Python utilizzato per individuare la correlazione dei dati relativi ai sensori del macchinario. Di seguito si riportano gli snippets di codice più rilevanti dello script.

```
def create_combinations(path_actuals, path_setpoints):
    array_files = []
    text_file_empty_files = open(os.path.join(position_path, 'Correlation', '
        empty_files.txt'), 'w')
    for file_name in glob.glob(os.path.join(position_path, path_actuals, '*.csv')):
        if os.stat(file_name).st_size != 0:
            array_files.append(file_name)
        else:
            text_file_empty_files.write(file_name + '\n')
    for file_name in glob.glob(os.path.join(position_path, path_setpoints, '*.csv')
        ):
        if os.stat(file_name).st_size != 0:
            array_files.append(file_name)
        else:
            text_file_empty_files.write(file_name + '\n')
    text_file_empty_files.close()
    combinations = list(itertools.combinations(array_files, 2))
    return combinations

def nearest(items, pivot):
    return min(items, key=lambda x: abs(x - pivot))

def sort_dataframe(small_dataset, big_dataset, start_index, time_delta):
    final_big_dataset = pd.DataFrame(columns = ['value_output','instant_time'])
    # For loop that iterates through the smallest dataframe starting from the index
         passed as a parameter to the function
    for index, row in islice(small_dataset.iterrows(), start_index, None):
        actual_instant_time = row['instant_time']
        output = big_dataset[(big_dataset['instant_time'] >= (actual_instant_time -
             pd.Timedelta(seconds = time_delta))) & (big_dataset['instant_time'] <=
            (actual_instant_time + pd.Timedelta(seconds = time_delta)))]
        if output.empty:
```

```
28            final_big_dataset = final_big_dataset.append(sort_dataframe(
                  small_dataset, big_dataset, index, (time_delta*2)))
29            break
30        else:
31            output = output.reset_index(drop = True)
32            nearest_item = nearest(output['instant_time'], actual_instant_time)
33            idx = output.index[output['instant_time'] == nearest_item]
34            final_big_dataset = final_big_dataset.append({'value_output': output['
                  value_output'].iloc[idx[0]], 'instant_time': output['instant_time'].
                  iloc[idx[0]]}, ignore_index=True)
35    return final_big_dataset
36
37 def calculate_correlation(combination):
38    correlated_variables = {
39      'pearson': '',
40      'spearman': ''
41    }
42    dataset1 = pd.read_csv(combination[0], sep=';', header = None)
43    dataset2 = pd.read_csv(combination[1], sep=';', header = None)
44    dataset1.columns = ['value_output', 'instant_time']
45    dataset2.columns = ['value_output', 'instant_time']
46    size_dataset1 = dataset1.shape[0]
47    size_dataset2 = dataset2.shape[0]
48    dataset1['instant_time'] = pd.to_datetime(dataset1.instant_time)
49    dataset2['instant_time'] = pd.to_datetime(dataset2.instant_time)
50    global_start_time = time.time()
51    time_delta = 1
52    start_index = 0
53    # Create directory for group by first dataset
54    Path(os.path.join(position_path, 'Correlation', os.path.basename(combination
          [0])[:-4])).mkdir(parents = True, exist_ok = True)
55    Path(os.path.join(position_path, 'Correlation', os.path.basename(combination
          [0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.path.basename(
          combination[1])[:-4])).mkdir(parents = True, exist_ok = True)
56    if size_dataset1 <= size_dataset2:
57        # Case where dataset 1 is smaller than dataset 2. Calling the "
              sort_dataframe()" function
```

```
58          final_big_dataset = sort_dataframe(dataset1, dataset2, start_index,
                time_delta)
59          name_dataset1 = os.path.basename(combination[0])[:-4]
60          name_final_big_dataset = os.path.basename(combination[1])[:-4]
61          # Retrieving the values in the "value_output" column of the two dataframes
                and converting them to numpy array
62          value_output_ds1 = dataset1.iloc[:,0:1].values
63          value_output_ds2 = final_big_dataset.iloc[:,0:1].values
64          # Create figure
65          fig = plt.figure(figsize=(15,8))
66          # Creation of boxplots
67          sns.boxplot(data = dataset1).set_title('Boxplot ' + str(os.path.basename(
                combination[0])[:-4]))
68          plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[0])
                [:-4] + '_boxplot.pdf'), dpi = 300)
69          plt.clf()
70          sns.boxplot(data = final_big_dataset).set_title('Boxplot ' + str(os.path.
                basename(combination[1])[:-4]))
71          plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[1])
                [:-4] + '_boxplot.pdf'), dpi = 300)
72          plt.clf()
73          # Filtering of duplicates
74          final_big_dataset = final_big_dataset.drop_duplicates(subset=['instant_time
                '], keep='last')
75          # Creation of the scatterplot
76          sns.lineplot(x = 'instant_time', y = 'value_output', data = dataset1,
                marker = 'o', alpha = .3, legend = 'brief', label = str(os.path.basename
                (combination[0])[:-4]), linestyle = '--').set_title('Scatterplot ' + os.
                path.basename(combination[0])[:-4] + ' - ' + os.path.basename(
                combination[1])[:-4])
77          sns.lineplot(x = 'instant_time', y = 'value_output', data =
                final_big_dataset, marker = 'o', alpha = .3, legend = 'brief', label =
                str(os.path.basename(combination[1])[:-4]))
78          plt.xticks(rotation=-45)
```

```
79          plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[0])
                [:-4] + '-' + os.path.basename(combination[1])[:-4] + '_scatterplot.pdf'
                ), dpi = 300)
80          plt.clf()
81          plt.close()
82      else:
83          # Case where dataset 2 is smaller than dataset 1. Calling the "
                sort_dataframe()" function
84          final_big_dataset = sort_dataframe(dataset2, dataset1, start_index,
                time_delta)
85          name_final_big_dataset = os.path.basename(combination[0])[:-4]
86          name_dataset2 = os.path.basename(combination[1])[:-4]
87          # Retrieving the values in the "value_output" column of the two dataframes
                and converting them to numpy array
88          value_output_ds1 = final_big_dataset.iloc[:,0:1].values
89          value_output_ds2 = dataset2.iloc[:,0:1].values
90          # Create figure
91          fig = plt.figure(figsize=(15,8))
92          # Creation of boxplots
93          sns.boxplot(data = final_big_dataset).set_title('Boxplot ' + str(os.path.
                basename(combination[0])[:-4]))
94          plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[0])
                [:-4] + '_boxplot.pdf'), dpi = 300)
95          plt.clf()
96          sns.boxplot(data = dataset2).set_title('Boxplot ' + str(os.path.basename(
                combination[1])[:-4]))
97          plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[1])
                [:-4] + '_boxplot.pdf'), dpi = 300)
98          plt.clf()
99          # Filtering of duplicates
100         final_big_dataset = final_big_dataset.drop_duplicates(subset=['instant_time
                '], keep='last')
```

```
101         # Creation of the scatterplot
102         sns.lineplot(x = 'instant_time', y = 'value_output', data =
                final_big_dataset, marker = 'o', alpha = .3, legend = 'brief', label =
                str(os.path.basename(combination[0])[:-4]), linestyle = '--').set_title(
                'Scatterplot ' + os.path.basename(combination[0])[:-4] + ' - ' + os.path
                .basename(combination[1])[:-4])
103         sns.lineplot(x = 'instant_time', y = 'value_output', data = dataset2,
                marker = 'o', alpha = .3, legend = 'brief', label = str(os.path.basename
                (combination[1])[:-4]))
104         plt.xticks(rotation=-45)
105         plt.savefig(os.path.join(position_path, 'Correlation', os.path.basename(
                combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.
                path.basename(combination[1])[:-4], os.path.basename(combination[0])
                [:-4] + '-' + os.path.basename(combination[1])[:-4] + '_scatterplot.pdf'
                ), dpi = 300)
106         plt.clf()
107         plt.close()
108     # Calculation of the Pearson index
109     pearsonr_start_time = time.time()
110     corr_pearson, _ = pearsonr(value_output_ds1.flatten(), value_output_ds2.flatten
            ())
111     # Calculation of the Spearman index
112     spearmanr_start_time = time.time()
113     corr_spearman, _ = spearmanr(value_output_ds1.flatten(), value_output_ds2.
            flatten())
114     # Write output on txt file
115     text_file = open(os.path.join(position_path, 'Correlation', os.path.basename(
            combination[0])[:-4], os.path.basename(combination[0])[:-4] + '-' + os.path
            .basename(combination[1])[:-4], os.path.basename(combination[0])[:-4] + '-'
             + os.path.basename(combination[1])[:-4] + '.txt'), 'w')
116     text_file.write('Seconds of pearsonr() execution: %s' % (time.time() -
            pearsonr_start_time))
117     text_file.write('\nPearsons correlation: %.3f' % corr_pearson)
118     rounded_corr_pearson = round(corr_pearson, 1)
119     if (math.isclose(rounded_corr_pearson, 1.0)):
120         text_file.write('\nPerfect positive relationship')
121         correlated_variables['pearson'] = '(>= 0.7) Pearson: ' + os.path.basename(
                combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
```

```
122     elif ((rounded_corr_pearson >= 0.7) and (rounded_corr_pearson < 1.0)):
123         text_file.write('\nStrong positive correlation')
124         correlated_variables['pearson'] = '(>= 0.7) Pearson: ' + os.path.basename(
                combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
125     elif ((rounded_corr_pearson > 0.0) and (rounded_corr_pearson < 0.7)):
126         text_file.write('\nPositive correlation')
127     elif math.isclose(rounded_corr_pearson, 0.0):
128         text_file.write('\nIndependent')
129     elif math.isclose(rounded_corr_pearson, -1.0):
130         text_file.write('\nPerfect negative relationship')
131         correlated_variables['pearson'] = '(<= -0.7) Pearson: ' + os.path.basename(
                combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
132     elif ((rounded_corr_pearson > -1.0) and (rounded_corr_pearson <= -0.7)):
133         text_file.write('\nStrong negative correlation')
134         correlated_variables['pearson'] = '(<= -0.7) Pearson: ' + os.path.basename(
                combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
135     elif ((rounded_corr_pearson < 0.0) and (rounded_corr_pearson > -0.7)):
136         text_file.write('\nNegative correlation')
137     else :
138         text_file.write('\nnan')
139     text_file.write('\n\nSeconds of spearmanr() execution: %s' % (time.time() -
            spearmanr_start_time))
140     text_file.write('\nSpearmans correlation: %.3f' % corr_spearman)
141     rounded_corr_spearman = round(corr_spearman, 1)
142     if (math.isclose(rounded_corr_spearman, 1.0)):
143         text_file.write('\nPerfect positive relationship')
144         correlated_variables['spearman'] = '(>= 0.7) Spearman: ' + os.path.basename
                (combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
145     elif ((rounded_corr_spearman >= 0.7) and (rounded_corr_spearman < 1.0)):
146         text_file.write('\nStrong positive correlation')
147         correlated_variables['spearman'] = '(>= 0.7) Spearman: ' + os.path.basename
                (combination[0])[:-4] + ' - ' + os.path.basename(combination[1])[:-4]
148     elif ((rounded_corr_spearman > 0.0) and (rounded_corr_spearman < 0.7)):
149         text_file.write('\nPositive correlation')
150     elif math.isclose(rounded_corr_spearman, 0.0):
151         text_file.write('\nIndependent')
152     elif math.isclose(rounded_corr_spearman, -1.0):
153         text_file.write('\nPerfect negative relationship')
```

```
154         correlated_variables['spearman'] = '(<= -0.7) Spearman: ' + os.path.
                basename(combination[0])[:-4] + ' - ' + os.path.basename(combination[1])
                [:-4]
155     elif ((rounded_corr_spearman > -1.0) and (rounded_corr_spearman <= -0.7)):
156         text_file.write('\nStrong negative correlation')
157         correlated_variables['spearman'] = '(<= -0.7) Spearman: ' + os.path.
                basename(combination[0])[:-4] + ' - ' + os.path.basename(combination[1])
                [:-4]
158     elif ((rounded_corr_spearman < 0.0) and (rounded_corr_spearman > -0.7)):
159         text_file.write('\nNegative correlation')
160     else :
161         text_file.write('\nnan')
162     text_file.write('\n\nSeconds of entire execution: %s' % (time.time() -
            global_start_time))
163     text_file.close()
164     return correlated_variables
165 
166 if __name__ == "__main__":
167     try:
168         threads_number = int(input('Insert the number of threads: '))
169     except ValueError:
170         print('The input is not a number')
171     try:
172         path_actuals = input("Enter the path to the actuals folder (e.g. 'path\\
                folder'): ")
173     except ValueError:
174         raise error
175     try:
176         path_setpoints = input("Enter the path to the setpoints folder (e.g. 'path
                \\folder'): ")
177     except ValueError:
178         raise error
179     with multiprocessing.Pool(initializer = worker_init, processes = threads_number
            ) as pool:
180         correlated_variables_total = pool.map(calculate_correlation,
                create_combinations(path_actuals, path_setpoints))
181     for correlated_variable in correlated_variables_total:
182         if correlated_variable.get('pearson') != '':
```

```
183             text_file_correlated_variables_pearson.write(correlated_variable.get('
                    pearson') + '\n')
184         if correlated_variable.get('spearman') != '':
185             text_file_correlated_variables_spearman.write(correlated_variable.get('
                    spearman') + '\n')
186     text_file_correlated_variables_pearson.close()
187     text_file_correlated_variables_spearman.close()
```

4.2.4 script_clustering.py

Il file contiene lo script Python utilizzato per eseguire il clustering dei dati relativi alle produzioni con guasto del macchinario. Di seguito si riportano gli snippets di codice più rilevanti dello script utilizzati per applicare la tecnica K-Means, le altre tecniche Agglomerative Clustering e DBSCAN sono state implementate in modo simile.

```
1  for file_name in glob.glob(os.path.join(path, '*.csv')):
2      Path(os.path.join(position_path, 'Clustering', os.path.basename(file_name)
           [:-4], 'Agglomerative Clustering')).mkdir(parents = True, exist_ok = True)
3      Path(os.path.join(position_path, 'Clustering', os.path.basename(file_name)
           [:-4], 'K-Means')).mkdir(parents = True, exist_ok = True)
4      Path(os.path.join(position_path, 'Clustering', os.path.basename(file_name)
           [:-4], 'DBSCAN')).mkdir(parents = True, exist_ok = True)
5      dataset = pd.read_csv(file_name, sep=';', header = None)
6      dataset.columns = ['item_id', 'start_time', 'end_time']
7      dataset['start_time'] = pd.to_datetime(dataset.start_time)
8      dataset['end_time'] = pd.to_datetime(dataset.end_time)
9      duration_series = pd.Series([])
10
11     for index, row in dataset.iterrows():
12         duration = pd.Series([pd.Timedelta(row['end_time'] - row['start_time']).
               seconds])
13         duration_series = duration_series.append([duration])
14
15     # Reindex the series
16     duration_series = duration_series.reset_index(drop = True)
17
18     # Adding "duration_time" column and removing other columns
```

```
    dataset['duration_time'] = duration_series
    dataset = dataset.drop('start_time', 1)
    dataset = dataset.drop('end_time', 1)

    # Integer One Hot Encode
    dataset['item_id'] = dataset['item_id'].astype(str)
    values = dataset['item_id'].to_numpy()
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(values)

    # Construction of the DataFrame to analyze
    dataset_to_analyze = pd.DataFrame()
    integer_encoded_reschaped = integer_encoded.ravel()
    integer_encoded_series = pd.Series(integer_encoded_reschaped)
    dataset_to_analyze['item_id'] = integer_encoded_series
    dataset_to_analyze['duration_time'] = dataset['duration_time']
    item_int_unique = np.unique(integer_encoded)

    # K-Means and Silhouette coefficient

    # Define dataset
    X = dataset_to_analyze.values

    # A list holds the silhouette coefficients for each k
    silhouette_coefficients = []

    # Create scatter plot for samples from each cluster
    fig = plt.figure(figsize = (20, 15))
    plt.style.use("fivethirtyeight")

    # Notice you start at 2 clusters for silhouette coefficient
    for k in range(start_k_range, end_k_range):
        # Define the model
        model = KMeans(n_clusters = k)

        # Fit the model
        model.fit(X)

```

```
57          # Assign a cluster to each example
58          yhat = model.predict(X)
59
60          # Retrieve unique clusters
61          clusters = unique(yhat)
62          score = silhouette_score(X, model.labels_)
63          silhouette_coefficients.append(score)
64
65          for cluster in clusters:
66              # Get row indexes for samples with this cluster
67              row_ix = where(yhat == cluster)
68
69              # Create scatter of these samples
70              pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
71
72          # Show the plot
73          plt.xticks(range(0, len(item_int_unique)))
74          plt.title("K-Means with " + str(k) + ' clusters')
75          plt.xlabel("Item description")
76          plt.ylabel("Breakdown duration (seconds)")
77          plt.tight_layout()
78          plt.savefig(os.path.join(position_path, 'Clustering', os.path.basename(
                file_name)[:-4], 'K-Means', 'k-means_' + str(k) + '.pdf'), dpi = 300)
79          plt.clf()
80
81      # Create scatter plot for Silhouette coefficient
82      fig = plt.figure(figsize = (20,10))
83      plt.style.use("fivethirtyeight")
84      plt.plot(range(start_k_range, end_k_range), silhouette_coefficients)
85      plt.xticks(range(start_k_range, end_k_range))
86      plt.title("Silhouette coefficient trend")
87      plt.xlabel("Number of clusters")
88      plt.ylabel("Silhouette coefficient")
89      plt.savefig(os.path.join(position_path, 'Clustering', os.path.basename(
            file_name)[:-4], 'K-Means', 'k-means_silhouette.pdf'), dpi = 300)
90      plt.clf()
91      plt.close()
```

4.2.5 script_dataframe_2_historical.py

Il file contiene lo script Python utilizzato per costruire il DataFrame 2 historical. Di seguito si riportano gli snippets di codice più rilevanti dello script. Come descritto nei capitoli relativi alla progettazione, gli script per costruire le altre quattro tipologie di DataFrame di training e di test sono delle varianti del presente script, implementate in modo simile.

```
1  def nearest(items, pivot):
2      return min(items, key = lambda x: abs(x - pivot))
3
4  def search_previous_production(df_productions_correct,
       production_breakdown_start_time, time_delta):
5      df_productions_correct['start_time'] = pd.to_datetime(df_productions_correct['
           start_time'])
6      df_productions_correct['end_time'] = pd.to_datetime(df_productions_correct['
           end_time'])
7      previous_production = pd.DataFrame(columns = ['start_time', 'end_time', '
           is_breakdown'])
8      output = df_productions_correct[(df_productions_correct['end_time'] >= (
           datetime.fromisoformat(production_breakdown_start_time) - pd.Timedelta(
           seconds = time_delta))) & (df_productions_correct['end_time'] <= datetime.
           fromisoformat(production_breakdown_start_time))]
9      # (time_delta < 100000000): maximum time_delta supported (3,17 years)
10     if ((output.empty) and (time_delta < 100000000)):
11             # If the output DataFrame is empty and time_delta is not too large, the
                   recursive function is invoked by doubling the time range
12             previous_production = previous_production.append(
                   search_previous_production(df_productions_correct,
                   production_breakdown_start_time, (time_delta*2)))
13     else:
14             if (time_delta < 100000000):
15                 # If the output DataFrame is not empty and time_delta is not too
                       large, its index is recreated
16                 output = output.reset_index(drop = True)
17                 # Calculation of the nearest instant of time among those in the
                       output DataFrame using the nearest() function
```

```
18              nearest_item = nearest(output['end_time'], datetime.fromisoformat(
                    production_breakdown_start_time))
19              # Index retrieval of the row with the closest timestamp
20              idx = output.index[output['end_time'] == nearest_item]
21              # Adding the row with the closest timestamp to previous_production
22              previous_production = previous_production.append({'start_time':
                    output['start_time'].iloc[idx[0]], 'end_time': output['end_time'
                    ].iloc[idx[0]], 'is_breakdown': output['is_breakdown'].iloc[idx
                    [0]]}, ignore_index = True)
23          else:
24              # Else insert a nan value (there aren't previous production
                    sufficiently near production_breakdown_start_time)
25              previous_production = previous_production.append({'start_time': '',
                    'end_time': '', 'is_breakdown': ''}, ignore_index = True)
26      return previous_production
27
28  def search_next_production(df_productions_correct, production_breakdown_end_time,
        time_delta):
29      df_productions_correct['start_time'] = pd.to_datetime(df_productions_correct['
            start_time'])
30      df_productions_correct['end_time'] = pd.to_datetime(df_productions_correct['
            end_time'])
31      next_production = pd.DataFrame(columns = ['start_time', 'end_time', '
            is_breakdown'])
32      output = df_productions_correct[(df_productions_correct['start_time'] <= (
            datetime.fromisoformat(production_breakdown_end_time) + pd.Timedelta(
            seconds = time_delta))) & (df_productions_correct['start_time'] >= datetime
            .fromisoformat(production_breakdown_end_time))]
33      if ((output.empty) and (time_delta < 100000000)):
34          next_production = next_production.append(search_next_production(
                df_productions_correct, production_breakdown_end_time, (time_delta
                *2)))
35      else:
36          if (time_delta < 100000000):
37              output = output.reset_index(drop = True)
38              nearest_item = nearest(output['start_time'], datetime.fromisoformat(
                    production_breakdown_end_time))
39              idx = output.index[output['start_time'] == nearest_item]
```

```
40              next_production = next_production.append({'start_time': output['
                    start_time'].iloc[idx[0]], 'end_time': output['end_time'].iloc[
                    idx[0]], 'is_breakdown': output['is_breakdown'].iloc[idx[0]]},
                    ignore_index = True)
41          else:
42              next_production = next_production.append({'start_time': nan, '
                    end_time': nan, 'is_breakdown': nan}, ignore_index = True)
43      return next_production
44
45  def get_other_productions_breakdown(count_new_breakdown):
46      global df_productions_breakdown_concat
47      df_other_productions_breakdown = pd.DataFrame(columns = ['start_time', '
            end_time', 'is_breakdown'])
48      df_productions_breakdown_concat = df_productions_breakdown_concat.sample(frac =
             1).reset_index(drop = True)
49      for index, row in df_productions_breakdown_concat.head(count_new_breakdown).
            iterrows():
50          df_other_productions_breakdown.loc[len(df_other_productions_breakdown)] = [
                row[1], row[2], 'true']
51      return df_other_productions_breakdown
52
53  def populate_df_productions_final(df_productions_breakdown_user):
54      global df_productions_final
55      count_new_breakdown = 0
56      for index_productions_breakdown_user, row_productions_breakdown_user in
            df_productions_breakdown_user.iterrows():
57          previous_production = search_previous_production(
                df_productions_correct_manipulated, row_productions_breakdown_user['
                start_time'], time_delta)
58          if (str(previous_production['start_time'].iloc[0]) != ''):
59              previous_of_previous_production = search_previous_production(
                    df_productions_correct_manipulated, str(previous_production['
                    start_time'].iloc[0]), time_delta)
60              next_production = search_next_production(
                    df_productions_correct_manipulated, row_productions_breakdown_user['
                    end_time'], time_delta)
61              # If the all the 3 rows are not already in the DataFrame append they to
                    df_productions_final
```

```
62            if not (((((df_productions_final['start_time']) == (previous_production[
                  'start_time'].iloc[0])) & ((df_productions_final['end_time']) == (
                  previous_production['end_time'].iloc[0]))).any()) and ((((
                  df_productions_final['start_time']) == (
                  previous_of_previous_production['start_time'].iloc[0])) & ((
                  df_productions_final['end_time']) == (
                  previous_of_previous_production['end_time'].iloc[0]))).any()) and
                  ((((df_productions_final['start_time']) == (next_production['
                  start_time'].iloc[0])) & ((df_productions_final['end_time']) == (
                  next_production['end_time'].iloc[0]))).any())):
63                previous_production.at[0, 'is_breakdown'] = 'true'
64                df_productions_final = df_productions_final.append(
                      previous_production)
65                df_productions_final = df_productions_final.append(
                      previous_of_previous_production)
66                df_productions_final = df_productions_final.append(next_production)
67            else:
68                count_new_breakdown = count_new_breakdown + 1
69        else:
70            count_new_breakdown = count_new_breakdown + 1
71    if (count_new_breakdown > 0):
72        df_other_productions_breakdown = get_other_productions_breakdown(
              count_new_breakdown)
73        populate_df_productions_final(df_other_productions_breakdown)
74
75 def search_sampling_in_range(df_sensor, production_final_start_time,
      production_final_end_time):
76    df_sensor[1] = pd.to_datetime(df_sensor[1])
77    output = df_sensor[(df_sensor[1] >= production_final_start_time) & (df_sensor
          [1] <= production_final_end_time)]
78    return output
79
80 text_file_empty_files = open(os.path.join(position_path, 'empty_files.txt'), 'w')
81 # Creation of CSV files lists
82 list_files_sensors = create_list_csv(path_sensors)
83 list_files_productions_breakdown = create_list_csv(path_productions_breakdown)
84 list_files_productions_correct = create_list_csv_productions_correct(
      path_productions_correct, check_dp_azione_02)
```

```
85 df_sensors = []
86 df_productions_breakdown = []
87 df_productions_correct = []
88 column_names_sensors = []
89 # Imports of all the CSV data in arrays of DataFrame
90 for file_sensor in list_files_sensors:
91     column_names_sensors.append(os.path.basename(file_sensor)[:-4])
92     df_sensors.append(pd.read_csv(file_sensor, sep = ';', header = None))
93 for file_productions_breakdown in list_files_productions_breakdown:
94     df_productions_breakdown.append(pd.read_csv(file_productions_breakdown, sep = '
           ;', header = None))
95 for file_productions_correct in list_files_productions_correct:
96     df_productions_correct.append(pd.read_csv(file_productions_correct, sep = ';',
           header = None))
97 # Concatenation of all the productions breakdown datasets
98 df_productions_breakdown_concat = pd.concat(df_productions_breakdown)
99 df_productions_breakdown_concat = df_productions_breakdown_concat.reset_index(drop
       = True)
100 # Concatenation of all the productions correct datasets
101 df_productions_correct_concat = pd.concat(df_productions_correct)
102 df_productions_correct_concat = df_productions_correct_concat.reset_index(drop =
       True)
103 # Drop all productions rows (breakdowns and corrects) that are older than the
       oldest sensor sampling or have a duration minor than 60 seconds
104 for index, row in df_productions_breakdown_concat.iterrows():
105     if ((datetime.fromisoformat(row[1]) < datetime.fromisoformat('2019-02-28
           13:06:39.297')) or ((datetime.fromisoformat(row[2]) - datetime.
           fromisoformat(row[1])).total_seconds() < 60)):
106         df_productions_breakdown_concat = df_productions_breakdown_concat.drop(
               index)
107 for index, row in df_productions_correct_concat.iterrows():
108     if ((datetime.fromisoformat(row[1]) < datetime.fromisoformat('2019-02-28
           13:06:39.297')) or ((datetime.fromisoformat(row[2]) - datetime.
           fromisoformat(row[1])).total_seconds() < 60)):
109         df_productions_correct_concat = df_productions_correct_concat.drop(index)
110 # Create productions breakdown DataFrame with n elements choosed by user inputs and
        manipulate the productions correct DataFrame
```

```
111 df_productions_breakdown_user = pd.DataFrame(columns = ['start_time', 'end_time', '
        is_breakdown'])
112 df_productions_correct_manipulated = pd.DataFrame(columns = ['start_time', '
        end_time', 'is_breakdown'])
113 # Shuffle the DataFrame that contains all the breakdowns
114 df_productions_breakdown_concat = df_productions_breakdown_concat.sample(frac = 1).
        reset_index(drop = True)
115 for index, row in df_productions_breakdown_concat.head(number_productions_breakdown
        ).iterrows():
116     df_productions_breakdown_user.loc[len(df_productions_breakdown_user)] = [row
            [1], row[2], 'true']
117 for index, row in df_productions_correct_concat.iterrows():
118     df_productions_correct_manipulated.loc[len(df_productions_correct_manipulated)]
            = [row[1], row[2], 'false']
119 # Creation of final DataFrame with correct productions near the breakdown
        productions (previous, previous of previous and next)
120 df_productions_final = pd.DataFrame(columns = ['start_time', 'end_time', '
        is_breakdown'])
121 populate_df_productions_final(df_productions_breakdown_user)
122 df_productions_final = df_productions_final.reset_index(drop = True)
123 column_names_final = []
124 for name_sensor in column_names_sensors:
125     column_names_final.append(name_sensor + '_short_mean')
126     column_names_final.append(name_sensor + '_short_std')
127     column_names_final.append(name_sensor + '_medium_mean')
128     column_names_final.append(name_sensor + '_medium_std')
129     column_names_final.append(name_sensor + '_long_mean')
130     column_names_final.append(name_sensor + '_long_std')
131 column_names_final.append('is_breakdown')
132 column_names_final.append('id_production')
133 df_final = pd.DataFrame(columns = column_names_final)
134 short = collections.deque(maxlen = 10)
135 medium = collections.deque(maxlen = 30)
136 long = collections.deque(maxlen = 60)
137 row = 0
138 count_index_sampling = 0
139 id_production = 0
```

```
for index_productions_final, row_productions_final in df_productions_final.iterrows():
    row = len(df_final)
    for index_sensors in range(len(df_sensors)):
        short.clear()
        medium.clear()
        long.clear()
        sampling_in_range = search_sampling_in_range(df_sensors[index_sensors], row_productions_final['start_time'], row_productions_final['end_time'])
        sampling_in_range = sampling_in_range.reset_index(drop = True)
        if (sampling_in_range.shape[0] >= 10):
            for index_sampling, row_sampling in sampling_in_range.iterrows():
                short.append(row_sampling[0])
                medium.append(row_sampling[0])
                long.append(row_sampling[0])
                if (index_sampling >= 9):
                    short_mean = np.mean(short)
                    short_std = np.std(short)
                    medium_mean = np.mean(medium)
                    medium_std = np.std(medium)
                    long_mean = np.mean(long)
                    long_std = np.std(long)
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 1]] = long_std
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 2]] = long_mean
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 3]] = medium_std
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 4]] = medium_mean
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 5]] = short_std
                    df_final.at[row + count_index_sampling, column_names_final[((index_sensors + 1) * 6) - 6]] = short_mean
                    df_final.at[row + count_index_sampling, 'is_breakdown'] = row_productions_final['is_breakdown']
                    df_final.at[row + count_index_sampling, 'id_production'] = id_production
```

```
168                     count_index_sampling = count_index_sampling + 1
169             count_index_sampling = 0
170         else:
171             print('sampling_in_range is too small')
172     id_production = id_production + 1
173 df_final.to_csv('df_final.csv', sep = ';', index = False, encoding = 'utf-8')
174 # Remove all the rows with is_breakdown = true and columns is_breakdown,
        id_production
175 df_final_cleared = df_final.drop(df_final[df_final.is_breakdown == 'true'].index)
176 df_final_cleared = df_final_cleared.drop('is_breakdown', 1)
177 df_final_cleared = df_final_cleared.drop('id_production', 1)
178 df_final_cleared.to_csv('df_final_cleared.csv', sep = ';', index = False, encoding
        = 'utf-8')
179 # Global means of all columns
180 df_final_global_means = df_final_cleared.mean(axis = 0)
181 # Array with all the id of the considered productions (all that have sufficient
        values)
182 unique_id_production = df_final['id_production'].drop_duplicates()
183 # Creation of the sub groups of the df_final, each group is composed of all the
        samplings of one production
184 array_sub_group = []
185 for id in unique_id_production:
186     df_final_sub_group = df_final.loc[df_final['id_production'] == id]
187     df_final_sub_group = df_final_sub_group.reset_index(drop = True)
188     df_final_sub_group.to_csv('folderprev\production_' + str(id) + '.csv', sep = ';
            ', index = False, encoding = 'utf-8')
189     # Slide all the columns of the sub group
190     for column_name, column in df_final_sub_group.iteritems():
191         # If the column is empty replace the NaN values with the global mean of
                that column, otherwise replace the NaN values with the local mean of
                that column
192         if ((column_name != 'is_breakdown') and (column_name != 'id_production')):
193             if column.dropna().empty:
194                 df_final_sub_group[column_name] = df_final_sub_group[column_name].
                        replace(np.nan, df_final_global_means[column_name], regex = True
                        )
195             else:
196                 column_mean = column.mean()
```

```
197                 df_final_sub_group[column_name] = df_final_sub_group[column_name].
                        replace(np.nan, column_mean, regex = True)
198     df_final_sub_group.to_csv('foldernext\production_' + str(id) + '.csv', sep = ';
            ', index = False, encoding = 'utf-8')
199     # Append the result to the array of sub groups
200     array_sub_group.append(df_final_sub_group)
201 # Concatenation of all the subgroups
202 df_final_concat = pd.concat(array_sub_group)
203 df_final_concat = df_final_concat.reset_index(drop = True)
204 df_final_concat.to_csv('df_final_concat_prima.csv', sep = ';', index = False,
        encoding = 'utf-8')
205 # Check if there are duplicated row, if is true remove they, keep only the firsts
        with precedence to is_breakdown = false
206 df_duplicates = df_final_concat[df_final_concat.duplicated(subset = df_final_concat
        .columns.tolist()[:-2], keep = False)]
207 if not df_duplicates.empty:
208     # Assign an id_group to each group of duplicated row
209     df_duplicates['id_group'] = df_duplicates.groupby(df_duplicates.columns.tolist
            ()[:-2], sort = False).ngroup() + 1
210     indexes_to_save = []
211     unique_id_group = df_duplicates['id_group'].drop_duplicates()
212     # Iterate through all the groups id
213     for id in unique_id_group:
214         # Create a temp dataframe for each group of duplicated row
215         df_duplicates_group = df_duplicates.loc[df_duplicates['id_group'] == id]
216         # If there is at least one row with is_breakdown = false, take the first
                line of these
217         if ((df_duplicates_group.is_breakdown == False).any()):
218             first_false_index = df_duplicates_group.index[df_duplicates_group['
                    is_breakdown'] == False]
219             indexes_to_save.append(first_false_index[0])
220         else:
221             # Else take the first row where is_breakdown = true
222             first_true_index = df_duplicates_group.index
223             indexes_to_save.append(first_true_index[0])
224     # Save all the index to delete
225     indexes_to_delete = np.setdiff1d(df_duplicates.index, indexes_to_save)
226     # Drop the row in df_final_concat
```

```
227     df_final_concat = df_final_concat.drop(indexes_to_delete)
228 # Drop id_production column
229 df_final_concat = df_final_concat.drop('id_production', 1)
230 df_final_concat = df_final_concat.reset_index(drop = True)
231 df_final_concat.to_csv('df_final_concat.csv', sep = ';', index = False, encoding =
        'utf-8')
232 # Standardization of df_final_concat:
233 column_is_breakdown = df_final_concat['is_breakdown']
234 df_final_concat = df_final_concat.drop('is_breakdown', 1)
235 # Get column names first
236 names = df_final_concat.columns
237 # Create the Scaler object
238 scaler = preprocessing.StandardScaler()
239 # Fit data on the scaler object
240 df_final_concat_standardized = scaler.fit_transform(df_final_concat)
241 # Export the scaler file to use in new data's dataframe
242 dump(scaler, 'scaler.bin', compress = True)
243 df_final_concat_standardized = pd.DataFrame(df_final_concat_standardized, columns =
         names)
244 df_final_concat_standardized['is_breakdown'] = column_is_breakdown
245 # Export CSV file of df_final_concat_standardized:
246 df_final_concat_standardized.to_csv('df_final_concat_standardized.csv', sep = ';',
        index = False, encoding = 'utf-8')
```

4.2.6 script_dataframe_5_nominal_sampling_distances.py

Il file contiene lo script Python utilizzato per costruire il DataFrame 5 nominal sampling distances. Di seguito si riportano gli snippets di codice più rilevanti dello script.

```
1 def count_correct_productions_previous(production_correct_start_time,
      previous_production_breakdown_end_time):
2     global df_productions_correct_concat_manipulated
3     df_productions_correct_concat_manipulated =
          df_productions_correct_concat_manipulated.reset_index(drop = True)
4     df_productions_correct_concat_manipulated['start_time'] = pd.to_datetime(
          df_productions_correct_concat_manipulated['start_time'])
```

```
5     df_productions_correct_concat_manipulated['end_time'] = pd.to_datetime(
          df_productions_correct_concat_manipulated['end_time'])
6     output = df_productions_correct_concat_manipulated[(
          df_productions_correct_concat_manipulated['end_time'] <= datetime.
          fromisoformat(production_correct_start_time)) & (
          df_productions_correct_concat_manipulated['start_time'] >=
          previous_production_breakdown_end_time[0])]
7     count_output_row = output.shape[0]
8     return count_output_row
9
10 def count_correct_productions_next(production_correct_end_time,
      next_production_breakdown_start_time):
11    global df_productions_correct_concat_manipulated
12    df_productions_correct_concat_manipulated =
          df_productions_correct_concat_manipulated.reset_index(drop = True)
13    df_productions_correct_concat_manipulated['start_time'] = pd.to_datetime(
          df_productions_correct_concat_manipulated['start_time'])
14    df_productions_correct_concat_manipulated['end_time'] = pd.to_datetime(
          df_productions_correct_concat_manipulated['end_time'])
15    output = df_productions_correct_concat_manipulated[(
          df_productions_correct_concat_manipulated['end_time'] <=
          next_production_breakdown_start_time[0]) & (
          df_productions_correct_concat_manipulated['start_time'] >= datetime.
          fromisoformat(production_correct_end_time))]
16    count_output_row = output.shape[0]
17    return count_output_row
18
19 def get_other_productions_correct(count_new_correct):
20    global df_productions_correct_concat
21    df_other_productions_correct = pd.DataFrame(columns = ['start_time', 'end_time'
          , 'is_breakdown'])
22    df_productions_correct_concat = df_productions_correct_concat.sample(frac = 1).
          reset_index(drop = True)
23    for index, row in df_productions_correct_concat.head(count_new_correct).
          iterrows():
24        df_other_productions_correct.loc[len(df_other_productions_correct)] = [row
              [1], row[2], 'false']
25    return df_other_productions_correct
```

```
26
27 def populate_df_productions_final(df_productions_correct_user):
28     global df_productions_final
29     count_new_correct = 0
30     for index_productions_correct_user, row_productions_correct_user in
           df_productions_correct_user.iterrows():
31         previous_production_breakdown = search_previous_production(
               df_productions_breakdown_manipulated, row_productions_correct_user['
               start_time'], time_delta)
32         next_production_breakdown = search_next_production(
               df_productions_breakdown_manipulated, row_productions_correct_user['
               end_time'], time_delta)
33         number_correct_productions_previous = count_correct_productions_previous(
               row_productions_correct_user['start_time'],
               previous_production_breakdown['end_time'])
34         number_correct_productions_next = count_correct_productions_next(
               row_productions_correct_user['end_time'], next_production_breakdown['
               start_time'])
35         # If there are at least n productions corrects in previous and next range
36         if ((number_correct_productions_previous >= treshold_distance) & (
               number_correct_productions_next >= treshold_distance)):
37             print('In execution...')
38             # If the row is not already in the DataFrame append they to
                   df_productions_final
39             if not ((((df_productions_final['start_time']) == (
                   row_productions_correct_user['start_time'])) & ((
                   df_productions_final['end_time']) == (row_productions_correct_user['
                   end_time']))).any()):
40                 df_productions_final = df_productions_final.append(
                       row_productions_correct_user)
41             else:
42                 count_new_correct = count_new_correct + 1
43         else:
44             count_new_correct = count_new_correct + 1
45     if (count_new_correct > 0):
46         df_other_productions_correct = get_other_productions_correct(
               count_new_correct)
47         populate_df_productions_final(df_other_productions_correct)
```

4.2.7 script_svm.py

Il file contiene lo script Python utilizzato per costruire e allenare il modello utilizzando Support Vector Machine. Di seguito si riportano gli snippets di codice più rilevanti dello script, nel quale si è utilizzata la libreria Scikit-learn. La versione dello script per il test del modello esportato è stata implementata in modo simile.

```
df = pd.read_csv(path_df, sep = ';')
# Delete empty rows
df = df.dropna()
df['is_breakdown'] = df['is_breakdown'].astype(str)
Y = []
target = df['is_breakdown']
for val in target:
    if(val == 'True'):
        Y.append(1)
    else:
        Y.append(-1)
X = df.drop('is_breakdown', 1)
# Creation of training and test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8)
# First test with standard parameters
clf = SVC(kernel = 'rbf', gamma = 'scale')
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
print('\nAccuracy: %.5f' % accuracy_score(y_test, y_pred))
print('\n')
print(classification_report(y_test, y_pred))
# Implementation of Grid Search
# Defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
# Fitting the model for grid search
grid.fit(x_train, y_train)
# Print best parameter after tuning
print('\nBest parameters found:')
```

```
print(grid.best_params_)
# Print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
# Re-run predictions and see classification report on this grid object
grid_predictions = grid.predict(x_test)
print('\nAccuracy: %.5f' % accuracy_score(y_test, grid_predictions))
print('\n')
print(classification_report(y_test, grid_predictions))
# Export the best model found
best_c = grid.best_params_['C']
best_gamma = grid.best_params_['gamma']
model = SVC(kernel = 'rbf', C = best_c, gamma = best_gamma)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('\nAccuracy: %.5f' % accuracy_score(y_test, y_pred))
print('\n')
print(classification_report(y_test, y_pred))
filename = 'model_svm.sav'
joblib.dump(model, filename)
```

4.2.8 script_cnn_1_classification.py

Il file contiene lo script Python utilizzato per costruire e allenare il modello utilizzando la Convolutional Neural Network di classificazione. Di seguito si riportano gli snippets di codice più rilevanti dello script, nel quale si è utilizzata la libreria Keras. La versione dello script per il test del modello esportato è stata implementata in modo simile.

```
df = pd.read_csv(path_df, sep = ';')
# Delete empty rows
df = df.dropna()
df['is_breakdown'] = df['is_breakdown'].astype(str)
Y = []
target = df['is_breakdown']
for val in target:
    if(val == 'True'):
        Y.append(1)
    else:
```

```
        Y.append(0)
X = df.drop('is_breakdown', 1)
# Creation of training and test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8)
# Conversions of DataFrames to numpy array and reshape them
x_train = x_train.to_numpy()
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.to_numpy()
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
y_train = np.array(y_train)
y_test = np.array(y_test)
# Build of the model
num_classes = len(np.unique(y_train))

def make_model(input_shape):
    input_layer = keras.layers.Input(input_shape)
    conv1 = keras.layers.Conv1D(filters = 124, kernel_size = 10, padding = "same")(
        input_layer)
    conv1 = keras.layers.BatchNormalization()(conv1)
    conv1 = keras.layers.ReLU()(conv1)
    conv2 = keras.layers.Conv1D(filters = 124, kernel_size = 10, padding = "same")(
        conv1)
    conv2 = keras.layers.BatchNormalization()(conv2)
    conv2 = keras.layers.ReLU()(conv2)
    conv3 = keras.layers.Conv1D(filters = 124, kernel_size = 10, padding = "same")(
        conv2)
    conv3 = keras.layers.BatchNormalization()(conv3)
    conv3 = keras.layers.ReLU()(conv3)
    gap = keras.layers.GlobalAveragePooling1D()(conv3)
    output_layer = keras.layers.Dense(num_classes, activation = "softmax")(gap)
    return keras.models.Model(inputs = input_layer, outputs = output_layer)

model = make_model(input_shape = x_train.shape[1:])
keras.utils.plot_model(model, show_shapes = True)
# Train the model
epochs = 100
batch_size = 30
callbacks = [
```

```
    keras.callbacks.ModelCheckpoint(
        # Export the best model file
        "model_cnn_1.h5", save_best_only = True, monitor = "val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor = "val_loss", factor = 0.5, patience = 20, min_lr = 0.0001
    ),
    keras.callbacks.EarlyStopping(monitor = "val_loss", patience = 50, verbose = 1)
        ,]
model.compile(
    optimizer = "adam",
    loss = "sparse_categorical_crossentropy",
    metrics = ["sparse_categorical_accuracy"],)
history = model.fit(
    x_train,
    y_train,
    batch_size = batch_size,
    epochs = epochs,
    callbacks = callbacks,
    validation_split = 0.2,
    verbose = 1,)
# Evaluate model on test data
model = keras.models.load_model("model_cnn_1.h5")
test_loss, test_acc = model.evaluate(x_test, y_test)
print("\nTest accuracy", test_acc)
print("Test loss", test_loss)
# Plot the model's training and validation loss
metric = "sparse_categorical_accuracy"
fig = plt.figure()
plt.plot(history.history[metric])
plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize = "large")
plt.xlabel("epoch", fontsize = "large")
plt.legend(["train", "val"], loc = "best")
fig.savefig('training_validation_loss.png', dpi = fig.dpi)
plt.close()
```

4.2.9 script_cnn_2_autoencoder.py

Il file contiene lo script Python utilizzato per costruire e allenare il modello utilizzando la Convolutional Neural Network con autoencoder. Di seguito si riportano gli snippets di codice più rilevanti dello script, nel quale si è utilizzata la libreria Keras.

```
df_training = pd.read_csv(path_df_training, sep = ';')
df_testing = pd.read_csv(path_df_testing, sep = ';')
df_training = df_training.drop('is_breakdown', 1)
df_testing = df_testing.drop('is_breakdown', 1)
# Create sequences
TIME_STEPS = 100
# Generated training sequences for use in the model

def create_sequences(values, time_steps = TIME_STEPS):
    output = []
    for i in range(len(values) - time_steps):
        output.append(values[i : (i + time_steps)])
    return np.stack(output)

x_train = create_sequences(df_training.values)
model = keras.Sequential(
    [
        layers.Input(shape = (x_train.shape[1], x_train.shape[2])),
        layers.Conv1D(filters = 64, kernel_size = 7, padding = "same", strides = 2,
             activation = "relu"),
        layers.Dropout(rate = 0.2),
        layers.Conv1D(filters = 32, kernel_size = 7, padding = "same", strides = 2,
             activation = "relu"),
        layers.Conv1DTranspose(filters = 32, kernel_size = 7, padding = "same",
            strides = 2, activation = "relu"),
        layers.Dropout(rate = 0.2),
        layers.Conv1DTranspose(filters = 64, kernel_size = 7, padding = "same",
            strides = 2, activation = "relu"),
        layers.Conv1DTranspose(filters = x_train.shape[2], kernel_size = 7, padding
             = "same"),
    ])
```

```
model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.001), loss = "mse")
model.summary()
# Train the model
history = model.fit(
    x_train,
    x_train,
    epochs = 10,
    batch_size = 30,
    validation_split = 0.1,
    callbacks=[
        keras.callbacks.EarlyStopping(monitor = "val_loss", patience = 5, mode = "min")
    ],)
fig_1 = plt.figure()
plt.plot(history.history["loss"], label = "Training Loss")
plt.plot(history.history["val_loss"], label = "Validation Loss")
plt.legend()
fig_1.savefig('training_validation_loss.png', dpi = fig_1.dpi)
# Detecting anomalies, get train MAE loss
x_train_pred = model.predict(x_train)
train_mae_loss = np.mean(np.abs(x_train_pred - x_train), axis = 1)
# Get reconstruction loss thresholds, the max number of each column
thresholds = train_mae_loss.max(axis = 0)
# Prepare test data, create sequences from test values
x_test = create_sequences(df_testing.values)
# Get test MAE loss
x_test_pred = model.predict(x_test)
test_mae_loss = np.mean(np.abs(x_test_pred - x_test), axis=1)
# Detect all the samples which are anomalies:
anomalies = 0
anomalies_indeces = []
for i in range(0, test_mae_loss.shape[0]):
    count = 0
    for j in range(0, test_mae_loss.shape[1]):
        if test_mae_loss[i,j] > thresholds[j]:
            count = count + 1
    if count >= 6:
```

```
63          anomalies = anomalies + 1
64          anomalies_indeces.append(i)
65  print("\nNumber of anomaly samples: ", anomalies)
66  print("Indices of anomaly samples: ", anomalies_indeces)
67  # Test the correctness of indeces
68  df_testing = pd.read_csv(path_df_testing, sep = ';')
69  df_testing['is_breakdown'] = df_testing['is_breakdown'].astype(str)
70  ture_positives = 0
71  false_positives = 0
72  for index in anomalies_indeces:
73      if df_testing.at[index, 'is_breakdown'] == 'True':
74          ture_positives = ture_positives + 1
75      else:
76          false_positives = false_positives + 1
77  print("\nTrue positives: ", ture_positives)
78  print("False positives: ", false_positives)
```

4.2.10 script_web_server.py

Il file contiene lo script Python utilizzato per costruire il web server con il framework Flask. Di seguito si riportano gli snippets di codice più rilevanti dello script.

```
1  app = Flask(__name__)
2  model = None
3  scaler = None
4  arrays_dictionary = {}
5  APP_ROOT = os.path.dirname(os.path.abspath(__file__))
6  UPLOAD_FOLD = 'uploads'
7  UPLOAD_FOLDER = os.path.join(APP_ROOT, UPLOAD_FOLD)
8  app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
9
10 # create_arrays_dictionary() function that creates arrays to be stored
11 def create_arrays_dictionary(number_sensors):
12     print('\ncreate_arrays_dictionary() called')
13     global arrays_dictionary
14     for i in range(1, number_sensors + 1, 1):
15         arrays_dictionary['sensor' + str(i) + '_short'] = collections.deque(maxlen
               = 10)
```

```
16        arrays_dictionary['sensor' + str(i) + '_medium'] = collections.deque(maxlen
              = 30)
17        arrays_dictionary['sensor' + str(i) + '_long'] = collections.deque(maxlen =
              60)
18
19 # load_file() function that loads model and scaler files in the script
20 def load_file(file_type, file_name):
21     print('\nload_file() called')
22     global model
23     global scaler
24     if file_type == 'model':
25         model = joblib.load(os.path.join('app', 'uploads', file_name))
26     if file_type == 'scaler':
27         scaler = joblib.load(os.path.join('app', 'uploads', file_name))
28
29 # index() function that redirect to uploader (index.html)
30 @app.route('/')
31 def index():
32    print('\nindex() called')
33    return render_template('index.html')
34
35 # set_number_sensors() function that creates the arrays for each sensor
36 @app.route('/numbersensors', methods = ['POST'])
37 def set_number_sensors():
38    if request.method == 'POST':
39       print('\nset_number_sensors() called')
40       create_arrays_dictionary(int(request.form['numbersensors']))
41       fill_arrays_test()
42       return render_template('index.html')
43
44 # upload_model() function that uploads model file to web server
45 @app.route('/modeluploader', methods = ['POST'])
46 def upload_model():
47    if request.method == 'POST':
48       print('\nupload_model() called')
49       f = request.files['file']
50       f.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(f.filename))
             )
```

```
51      load_file('model', f.filename)
52      return render_template('index.html')
53
54  # upload_scaler() function that uploads scaler file to web server
55  @app.route('/scaleruploader', methods = ['POST'])
56  def upload_scaler():
57     if request.method == 'POST':
58        print('\nupload_scaler() called')
59        f = request.files['file']
60        f.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(f.filename))
              )
61        load_file('scaler', f.filename)
62        return render_template('index.html')
63
64  # get_data() function is the endpoint that receive data from POST end predict
      result
65  @app.route('/getdata/', methods = ['POST'])
66  def get_data():
67     if request.method == 'POST':
68        print('\nget_data() called')
69        global model
70        global scaler
71        # Get sensor data from POST request and save them in a dictionary
72        sensors_dictionary = {}
73        array_row = np.array([])
74        for i in range(1, int(request.form.get('nsensors')) + 1, 1):
75           sensors_dictionary['sensor' + str(i)] = request.form.get('sensor' + str(
                 i))
76        # Build the array of values, this is our new record (row of DataFrame)
77        for key, value in sensors_dictionary.items():
78           arrays_dictionary[key + '_short'].append(float(value))
79           arrays_dictionary[key + '_medium'].append(float(value))
80           arrays_dictionary[key + '_long'].append(float(value))
81           array_row = np.append(array_row, [np.mean(arrays_dictionary[key + '
                 _short'])])
82           array_row = np.append(array_row, [np.std(arrays_dictionary[key + '_short
                 '])])
```

```
83            array_row = np.append(array_row, [np.mean(arrays_dictionary[key + '
                  _medium'])])
84            array_row = np.append(array_row, [np.std(arrays_dictionary[key + '
                  _medium'])])
85            array_row = np.append(array_row, [np.mean(arrays_dictionary[key + '_long
                  '])])
86            array_row = np.append(array_row, [np.std(arrays_dictionary[key + '_long'
                  ])])
87        # Fit your data on the scaler object for standardization
88        array_row_standardized = scaler.transform(array_row.reshape(1, -1))
89        # Predict y_pred using the loaded trained model:
90        y_pred = model.predict(array_row_standardized)
91        if y_pred[0] == 1:
92            result = 'Production with breakdown (1)'
93        else:
94            result = 'Production correct (-1)'
95        print('\nPredicted output:')
96        print(result)
97        return 'Data sent correctly! ' + result
```

4.2.11 configuration_page.html

Il file contiene il codice HTML utilizzato per costruire la pagina di configurazione del sistema, anche tramite CSS e Bootstrap. Di seguito si riportano gli snippets di codice più rilevanti della pagina.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8"/>
5   <title>Manutenzione predittiva</title>
6   <meta name="description" content="Manutenzione predittiva"/>
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap
        .min.css" rel="stylesheet" integrity="sha384-giJF6kkoqNQ00vy+
        HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1" crossorigin="anonymous">
8   <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='
        styles/default.css') }}">
```

```
9   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome
        /4.7.0/css/font-awesome.min.css">
10 </head>
11   <body>
12     <div class="position-relative">
13       <div class="position-absolute top-0 start-0 w-100 toolbar row">
14         <div class="col-1"></div>
15         <div class="col">
16           <center><h4><i class="fa fa-cog"></i> Configurazione del sistema di
                manutenzione predittiva</h4></center>
17         </div>
18         <div class="col-1"></div>
19       </div>
20     </div>
21     <div class="container">
22       <div class="row">
23         <div class="col-12 m-auto">
24           <div class="row mt-5">
25             <div class="col">
26               <br>
27               <center><h5><i class="fa fa-tachometer"></i> Inserisci il numero dei
                    sensori:</h5></center>
28               <br>
29             </div>
30           </div>
31           <div class="row">
32             <form action="https://opera-predictive-maintenance.herokuapp.com/
                    numbersensors" method="POST" enctype="multipart/form-data">
33               <div class="input-group mb-3">
34                 <input type="number" class="form-control" name="numbersensors" min="
                        1" max="1000">
35               </div>
36               <button type="submit" class="btn btn-primary w-100">Invia il numero</
                    button>
37             </form>
38           </div>
39           <br>
40           <hr>
```

```
41        </div>
42      </div>
43      <div class="row">
44        <div class="col-6 m-auto">
45          <div class="row mt-5">
46            <div class="col">
47              <center><h5><i class="fa fa-upload"></i> Carica il file del modello:</
                    h5></center>
48              <br>
49              <center><p>Esempio: "model.sav"</p></center>
50            </div>
51          </div>
52          <div class="row">
53            <form action="https://opera-predictive-maintenance.herokuapp.com/
                    modeluploader" method="POST" enctype="multipart/form-data">
54              <div class="input-group mb-3">
55                <input type="file" class="form-control" name="file"/>
56              </div>
57              <button type="submit" class="btn btn-primary w-100">Carica il file</
                    button>
58            </form>
59          </div>
60        </div>
61        <div class="col-6 m-auto">
62          <div class="row mt-5">
63            <div class="col">
64              <center><h5><i class="fa fa-upload"></i> Carica il file dello scaler:<
                    /h5></center>
65              <br>
66              <center><p>Esempio: "scaler.bin"</p></center>
67            </div>
68          </div>
69          <div class="row">
70            <form action="https://opera-predictive-maintenance.herokuapp.com/
                    scaleruploader" method="POST" enctype="multipart/form-data">
71              <div class="input-group mb-3">
72                <input type="file" class="form-control" name="file"/>
73              </div>
```

```
74              <button type="submit" class="btn btn-primary w-100">Carica il file</
                    button>
75            </form>
76          </div>
77        </div>
78      </div>
79      <br>
80      <hr>
81      <center><p>Copyright c 2020 - 2021 Open Data. Tutti i diritti riservati.</p><
            /center>
82    </div>
83  </body>
84 </html>
```

Capitolo 5

Validazione

Nei capitoli precedenti si sono descritte le fasi di progettazione e implementazione del sistema di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione. L'obiettivo di questo capitolo è invece quello di approfondire la fase di validazione del progetto, descrivendo gli ultimi due step del modello CRISP-DM, cioè Evaluation e Deployment. In queste ultime due fasi si sono valutati i risultati ottenuti dai modelli di Machine Learning e Deep Learning costruiti, selezionando e valutando in modo più approfondito quello che presenta la migliore performance. Per quanto riguarda il rilascio del progetto, si è realizzato un web server per eseguire il sistema sul cloud computing e sull'edge computing. Infine si è svolta un'analisi sul modo in cui tale sistema debba essere integrato in Opera MES.

5.1 Evaluation

In questa quinta fase del modello CRISP-DM avviene la valutazione dei risultati ottenuti dai modelli di Machine Learning e Deep Learning costruiti. Mentre la valutazione di tali modelli durante la precedente fase di Modeling si svolgeva da un punto di vista tecnico, ad esempio per regolare i parametri degli algoritmi utilizzati, la fase di Evaluation esamina in modo più ampio quale modello soddisfa meglio gli obiettivi del business e come procedere successivamente. Questa fase è composta dai seguenti task:

- **Valutazione dei risultati:** si valuta se i modelli soddisfano i criteri di successo aziendale, scegliendo quello o quelli che possono essere approvati per un'effettiva applicazione.
- **Revisione del processo:** si controlla il lavoro svolto per valutare se è stato trascurato qualche passaggio o se alcuni task non sono stati eseguiti correttamente. Si riassumono i risultati ottenuti, correggendo se necessario eventuali errori.
- **Determinazione dei passaggi successivi:** basandosi sui risultati dei task precedenti, si determina se procedere con la distribuzione del sistema, eseguire ulteriori iterazioni, oppure avviare nuovi progetti.

5.1.1 Test dei DataFrame sui modelli e risultati

Inizialmente, i modelli di Machine Learning e Deep Learning costruiti, sono stati allenati e testati con i DataFrame realizzati a partire dai *dati originali* esportati dal database. Successivamente, i tre modelli costruiti con SVM e le due CNN sono stati testati con i *dati nuovi* e i cosiddetti *dati mix*. Di seguito è descritto cosa si intende con i termini dati originali, dati nuovi e dati mix:

- **Dati originali:** si tratta dei dati registrati dal macchinario industriale dal mese di febbraio 2019 fino al mese di ottobre 2020. Sono i dati utilizzati per eseguire il training dei modelli e il loro testing iniziale.
- **Dati nuovi:** si tratta dei dati registrati dal macchinario industriale durante i mesi di novembre e dicembre 2020. Sono i dati utilizzati per eseguire un ulteriore testing dei modelli utilizzando record completamente nuovi.
- **Dati mix:** si tratta di un mix di dati originali e di dati nuovi. Sono i dati utilizzati per eseguire ulteriori verifiche, in particolare un nuovo training e successivo testing di alcuni dei modelli costruiti.

Tutti i DataFrame delle varie tipologie creati per essere utilizzati come test set, sono stati standardizzati utilizzando i file *scaler.bin* esportati in precedenza durante la creazione dei relativi training set. I dati nuovi sono stati esportati dal database Microsoft SQL

Support Vector Machine			
DF training set e test set	**Training set**	**Test set**	**Accuracy**
DataFrame 1 n.	Dati originali	Dati originali	72,50%
	Dati originali	Dati nuovi	73,81%
DataFrame 2 h.	Dati originali	Dati originali	99,85%
	Dati originali	Dati nuovi	79,01%
DataFrame 3 n.s.	Dati originali	Dati originali	93,97%
	Dati originali	Dati nuovi	74,88%
	Dati mix	Dati mix	97,07%
	Dati mix	Dati nuovi	66,19%

Tabella 5.1: Risultati del modello allenato con SVM

Server eseguendo di nuovo le query SQL progettate inizialmente in fase di Data Understanding. Nella tabella 5.1 sono riportati i risultati di accuracy ottenuti utilizzando il modello creato tramite **Support Vector Machine**. Come si può vedere dalla tabella, tale modello è stato utilizzato con DataFrame 1 nominal, DataFrame 2 historical e DataFrame 3 nominal sampling. Tutte le tipologie di dataset sono state utilizzate per allenare e testare il modello con i dati originali. Successivamente, il modello allenato con i dati originali è stato testato nuovamente, utilizzando però le tre tipologie di dataset create a partire dai dati nuovi. In seguito, il modello è stato allenato e testato nuovamente utilizzando il DataFrame 3 nominal sampling costruito a partire dai dati mix. Infine, si è eseguito un ulteriore test della stessa tipologia di dataset ma costruito a partire dai dati nuovi sul modello allenato con i dati mix. I risultati dell'accuracy ottenuti utilizzando i DataFrame costruiti a partire dai dati originali e mix sono molto alti, spesso oltre il 90%, in particolare il DataFrame 2 historical con tali dati presenta overfitting. Utilizzando invece i dati nuovi per il testing i risultati dell'accuracy calano, rimanendo comunque su buoni livelli, intorno al 70% - 80%. Inoltre, nel caso dei dati nuovi testati sul modello allenato con i dati mix, l'accuracy scende ulteriormente al 66%.

Convolutional Neural Network di classificazione			
DF training set e test set	**Training set**	**Test set**	**Accuracy**
DataFrame 1 n.	Dati originali	Dati originali	71,49%
	Dati originali	Dati nuovi	73,81%
DataFrame 2 h.	Dati originali	Dati originali	99,73%
	Dati originali	Dati nuovi	66,68%
DataFrame 3 n.s.	Dati originali	Dati originali	93,86%
	Dati originali	Dati nuovi	45,46%
	Dati mix	Dati mix	93,35%
	Dati mix	Dati nuovi	59,21%

Tabella 5.2: Risultati del modello allenato con la CNN di classificazione

Nella tabella 5.2 sono riportati i risultati di accuracy ottenuti utilizzando il modello creato tramite la **Convolutional Neural Network di classificazione**. Il procedimento di valutazione è stato uguale a quello appena descritto per Support Vector Machine. Come si può vedere dalla tabella, tale modello è stato utilizzato con DataFrame 1 nominal, DataFrame 2 historical e DataFrame 3 nominal sampling. Tutte le tipologie di dataset sono state utilizzate per allenare e testare il modello con i dati originali. Successivamente, il modello allenato con i dati originali è stato testato nuovamente, utilizzando però le tre tipologie di dataset create a partire dai dati nuovi. In seguito, il modello è stato allenato e testato nuovamente utilizzando il DataFrame 3 nominal sampling costruito a partire dai dati mix. Infine, si è eseguito un ulteriore test della stessa tipologia di dataset ma costruito a partire dai dati nuovi sul modello allenato con i dati mix. I risultati dell'accuracy ottenuti utilizzando i DataFrame costruiti a partire dai dati originali e mix sono molto alti, spesso oltre il 90%, in particolare il DataFrame 2 historical con tali dati presenta overfitting. Utilizzando invece i dati nuovi per il testing i risultati dell'accuracy si riducono notevolmente, attestandosi intorno al 45% - 66%. Invece, nel caso del DataFrame 1 nominal con i dati nuovi testati sul modello allenato con i dati originali, l'accuracy rimane comunque su buoni livelli intorno al 73%.

Convolutional Neural Network con autoencoder					
Col.	**DF training set**	**DF test set**	**Training set**	**Test set**	**TP**
2	DataFrame 4 n.s.v.	DataFrame 3 n.s.	Dati originali	Dati originali	33,1%
			Dati originali	Dati nuovi	19,1%
6	DataFrame 5 n.s.d.	DataFrame 3 n.s.	Dati originali	Dati originali	35,6%
			Dati originali	Dati nuovi	10,6%

Tabella 5.3: Risultati del modello allenato con la CNN con autoencoder

Infine, nella tabella 5.3 sono riportati i risultati di performance ottenuti utilizzando il modello creato tramite la **Convolutional Neural Network con autoencoder**. Il procedimento di valutazione è stato diverso rispetto ai due modelli precedenti. Come si può vedere dalla tabella, tale modello è stato utilizzato con il DataFrame 4 nominal sampling variant e il DataFrame 5 nominal sampling distances. Tali tipologie di dataset sono state utilizzate per allenare il modello con i dati originali. Successivamente, tale modello è stato testato utilizzando il DataFrame 3 nominal sampling creato a partire dai dati originali e in seguito quello creato a partire dai dati nuovi. L'utilizzo di due tipologie di DataFrame diverse per il training set e il test set è dovuto all'approccio stesso che utilizza la rete neurale con autoencoder. In questo modo i dataset di training contengono solo dati di produzioni corrette, mentre i dataset di testing contengono sia produzioni corrette che produzioni con guasto. Il numero n di sensori (quindi di colonne) che devono superare la soglia impostata per classificare il relativo record come guasto è stato scelto dopo diversi test. In particolare i risultati migliori si sono registrati impostando $n=2$ nel caso del DataFrame 4 nominal sampling variant e $n=6$ nel caso del DataFrame 5 nominal sampling distances. Tutti i dati riportati si riferiscono ai true positives. I risultati della percentuale di record associati a guasti che sono stati classificati correttamente utilizzando i DataFrame costruiti a partire dai dati originali sono bassi, intorno al 35%. Utilizzando i DataFrame costruiti a partire dai dati nuovi i risultati calano ulteriormente, attestandosi intorno al 10% - 20%. Da evidenziare inoltre che in tutti i casi il numero dei false positives rilevati è considerevole.

Confrontando i risultati ottenuti utilizzando i tre modelli creati, si nota come la seconda Convolutional Neural Network con autoencoder presenti i risultati peggiori. Questo è dovuto probabilmente al problema descritto nei precedenti capitoli, relativo ai finti guasti che in realtà sono dei fermi macchina. Tale tipologia di rete neurale risulta essere più sensibile a questi dati errati che sono presenti nel dataset e che si è tentato inizialmente di rimuovere con un filtro. Anche i due approcci utilizzati per la costruzione del DataFrame 4 nominal sampling variant e del DataFrame 5 nominal sampling distances non sono riusciti a mitigare a sufficienza il problema dei finiti guasti. Si ricorda che il DataFrame 4 considera come corrette le produzioni immediatamente successive al guasto, mentre il DataFrame 5 considera come corrette le produzioni che si trovano ad una certa distanza da un guasto, in avanti e indietro nel tempo. Per quanto riguarda la prima Convolutional Neural Network di classificazione, essa presenta risultati migliori rispetto all'altra rete neurale, ma comunque non molto buoni nei test con i dati nuovi. Questa rete risulta essere quindi meno sensibile della precedente al problema dei finti guasti. Infine, il modello costruito utilizzando Support Vector Machine presenta i risultati migliori, specialmente nel test con i dati nuovi, mentre con i dati originali in alcuni casi tende all'overfitting. Si nota quindi che, nel contesto del progetto realizzato, SVM raggiunge risultati migliori rispetto alle due CNN, pur essendo apparentemente una tecnica meno potente. In conclusione quindi, si è selezionato come migliore il modello costruito con Support Vector Machine usando il DataFrame 2 historical, poiché raggiunge un'accuracy di quasi l'80% con i dati nuovi. Di tale modello si è realizzata una valutazione più approfondita, descritta nel seguente capitolo.

5.1.2 Valutazione del modello selezionato e risultati

Come descritto nel capitolo precedente, il modello costruito con Support Vector Machine e il DataFrame 2 historical è stato scelto come modello migliore, in quanto presenta il più alto livello di accuracy. Tale modello verrà quindi utilizzato nella fase successiva di Deployment, eseguendolo nel web server e caricandolo sul cloud computing. Per eseguire la valutazione del modello migliore selezionato, esso è stato allenato utilizzando il DataFrame 2 historical costruito a partire dai dati originali come training set. Successi-

vamente, tale modello è stato testato con tre differenti test set, standardizzati utilizzando il file *scaler.bin* esportato in precedenza durante la creazione del training set:

- **Dati originali:** il primo test si è eseguito utilizzando il DataFrame 2 historical costruito a partire dai dati originali, utilizzando una porzione degli stessi che fosse diversa dai dati originali utilizzati per il training.
- **Dati nuovi:** il secondo test si è eseguito utilizzando il DataFrame 2 historical costruito a partire dai dati nuovi.
- **Dati mix:** il terzo test si è eseguito utilizzando il DataFrame 2 historical costruito a partire da un mix di dati originali e dati nuovi.

Tutti i tre test eseguiti sono stati valutati utilizzando le metriche di *accuracy*, *precision*, *recall* e *f1-score*, oltre alla realizzazione delle *confusion matrix*. Tali metriche utilizzano i differenti tipi di risultati prodotti dal modello, cioè *true positive*, *true negative*, *false positive* e *false negative*. Di seguito vengono definite tutte le metriche appena citate:

- **True positive (TP):** è il numero di veri positivi. Cioè i campioni classificati correttamente dal modello come *true* che erano effettivamente *true*.
- **True negative (TN):** è il numero di veri negativi. Cioè i campioni classificati correttamente dal modello come *false* che erano effettivamente *false*.
- **False positive (FP):** è il numero di falsi positivi. Cioè i campioni classificati erroneamente dal modello come *true* che invece erano *false*.
- **False negative (FN):** è il numero di falsi negativi. Cioè i campioni classificati erroneamente dal modello come *false* che invece erano *true*.
- **Accuracy:** è la percentuale di classificazioni corrette. Viene calcolata come il rapporto delle previsioni corrette sul numero totale di previsioni, con la formula (TP+TN) / (TP+TN+FP+FN).
- **Precision:** è la percentuale di classificazioni positive che sono corrette. Viene calcolata come il rapporto tra i true positive e la somma dei true positive e false positive, con la formula *TP / (TP+FP)*.

- **Recall:** è la percentuale di elementi positivi del test set che sono stati classificati come positivi. Viene calcolata come il rapporto tra i true positive e la somma dei true positive e false negative, con la formula *TP / (TP+FN)*.

- **F1-score:** è la media ponderata di precision e recall, dove il punteggio migliore è 1 e il punteggio peggiore è 0. Viene calcolata con la formula *2 * (precision * recall) / (precision + recall)*.

- **Confusion matrix:** è una rappresentazione dell'accuratezza di classificazione del modello, le colonne della matrice indicano i valori predetti e le righe indicano i valori reali.

In tutti i tre test svolti, la *label 1* indica i campionamenti associati a produzioni con guasto (cioè con la colonna is_breakdown settata a true), mentre la *label -1* indica i campionamenti associati a produzioni corrette, (cioè con la colonna is_breakdown settata a false). Nel **primo test** eseguito con il test set di dati originali, come si può vedere dalla confusion matrix della figura 5.1, il modello riesce a predire 2963 produzioni corrette con successo, mentre soltanto una produzione corretta è stata classificata in modo errato. Per quanto riguarda invece le produzioni con guasto, il modello riesce a predirne 1094 con successo, mentre soltanto una è stata classificata in modo errato. Di seguito alcune osservazioni sui risultati delle metriche mostrati nella figura 5.2:

- **Accuracy:** l'accuracy del modello è del 99.9%, si è quindi in presenza di overfitting.

- **Precision:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 1.00 *(2963 / (2963 + 1) = 0.999)*. Per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato sempre di 1.00 *(1094 / (1094 + 1) = 0.999)*.

- **Recall:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 1.00 *(2963 / (2963 + 1) = 0.999)*. Per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato sempre di 1.00 *(1094 / (1094 + 1) = 0.999)*.

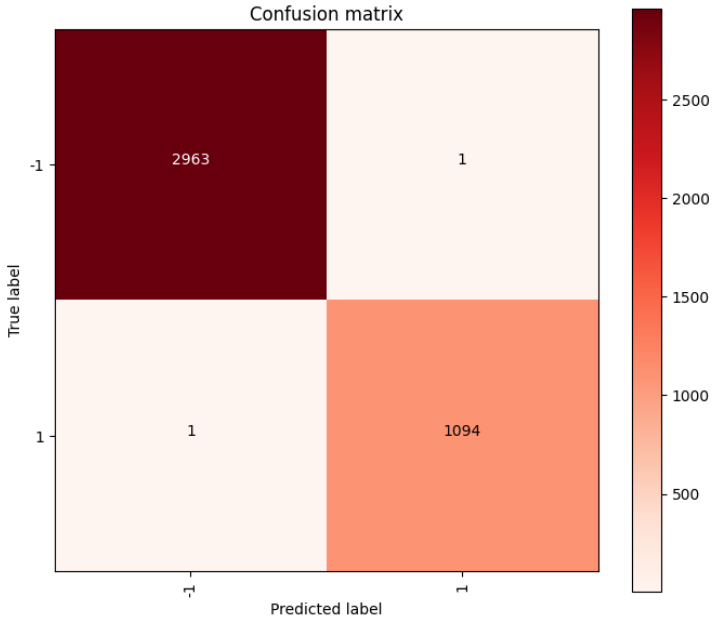

Figura 5.1: Confusion matrix del primo test con i dati originali

Accuracy: 0.99951

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	2964
1	1.00	1.00	1.00	1095
accuracy			1.00	4059
macro avg	1.00	1.00	1.00	4059
weighted avg	1.00	1.00	1.00	4059

Figura 5.2: Risultati del primo test con i dati originali

Nel **secondo test** eseguito con il test set di dati nuovi, come si può vedere dalla confusion matrix della figura 5.3, il modello riesce a predire 11057 produzioni corrette con successo, mentre 146 produzioni corrette sono state classificate in modo errato. Per quanto riguarda invece le produzioni con guasto, il modello riesce a predirne 0 con successo, mentre 2730 sono state classificate in modo errato. Questo poiché, come verrà descritto successivamente, nei dati nuovi raccolti nei mesi di novembre e dicembre 2020 è presen-

te un elevato numero di falsi guasti. Oltre al fatto che, come confermato dal referente dell'azienda cliente STS Tecnopolimeri, non sono stati riscontrati guasti veri, in quanto molte produzioni in questo periodo erano già ampiamente conosciute. Di seguito alcune osservazioni sui risultati delle metriche mostrati nella figura 5.4:

- **Accuracy:** l'accuracy del modello è del 79.4%.
- **Precision:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 0.80 *(11057 / (11057 + 2730) = 0.802)*. Mentre per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato di 0.00 *(0 / (0 + 146) = 0)*.
- **Recall:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 0.99 *(11057 / (11057 + 146) = 0.987)*. Mentre per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato di 0.00 *(0 / (0 + 2730) = 0)*.

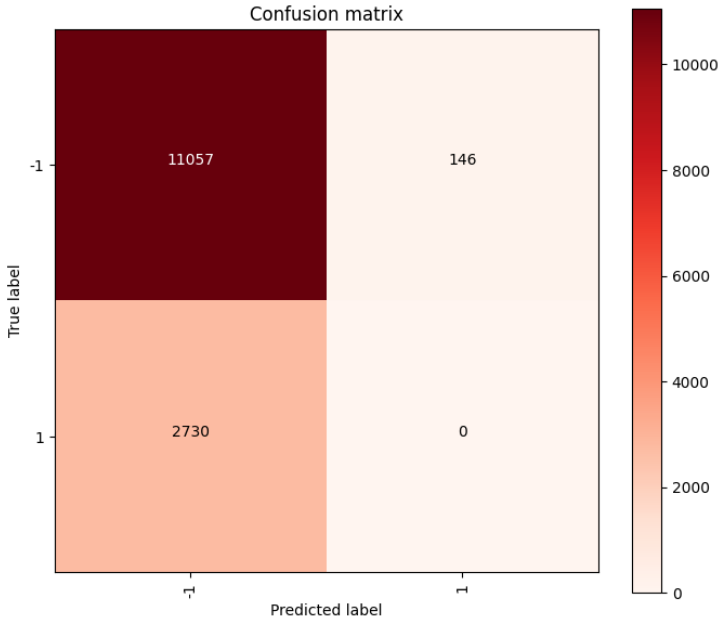

Figura 5.3: Confusion matrix del secondo test con i dati nuovi

```
Accuracy: 0.79358

              precision    recall  f1-score   support

          -1       0.80      0.99      0.88     11203
           1       0.00      0.00      0.00      2730

    accuracy                           0.79     13933
   macro avg       0.40      0.49      0.44     13933
weighted avg       0.64      0.79      0.71     13933
```

Figura 5.4: Risultati del secondo test con i dati nuovi

Infine, nel **terzo test** eseguito con il test set di dati mix, come si può vedere dalla confusion matrix della figura 5.5, il modello riesce a predire 14020 produzioni corrette con successo, mentre 147 produzioni corrette sono state classificate in modo errato. Per quanto riguarda invece le produzioni con guasto, il modello riesce a predirne 1094 con successo, mentre 2731 sono state classificate in modo errato. Questo avviene sempre per le motivazioni descritte precedentemente e relative al problema dei finti guasti. Di seguito alcune osservazioni sui risultati delle metriche mostrati nella figura 5.6:

- **Accuracy:** l'accuracy del modello è dell'84%.
- **Precision:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 0.84 *(14020 / (14020 + 2731) = 0.837)*. Mentre per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato di 0.88 *(1094 / (1094 + 147) = 0.882)*.
- **Recall:** per i campionamenti associati a produzioni corrette si ottiene un risultato arrotondato di 0.99 *(14020 / (14020 + 147) = 0.989)*. Mentre per i campionamenti associati a produzioni con guasto si ottiene un risultato arrotondato di 0.29 *(1094 / (1094 + 2731) = 0.286)*.

Confrontando i risultati ottenuti dai tre test eseguiti si conferma e si approfondisce quanto già riscontrato nel precedente task. Cioè che il primo test, eseguito con il test set dei dati originali, presenta overfitting. Nonostante ciò bisogna considerare che il dominio del problema al quale si applica il modello costruito, cioè la manutenzione predittiva di

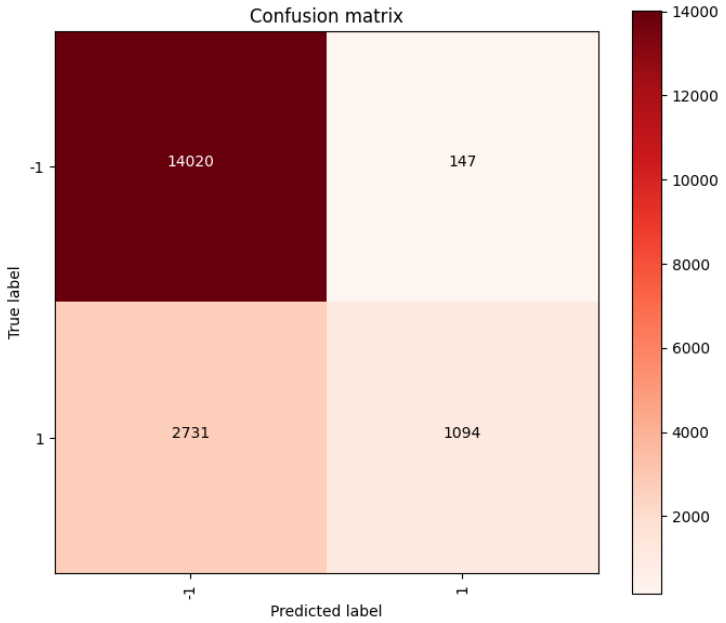

Figura 5.5: Confusion matrix del terzo test con i dati mix

Accuracy: 0.84004

	precision	recall	f1-score	support
-1	0.84	0.99	0.91	14167
1	0.88	0.29	0.43	3825
accuracy			0.84	17992
macro avg	0.86	0.64	0.67	17992
weighted avg	0.85	0.84	0.81	17992

Figura 5.6: Risultati del terzo test con i dati mix

macchinari industriali per lo stampaggio plastico a iniezione, non implica solitamente un frequente cambiamento degli articoli prodotti. Di conseguenza il modello, almeno nel breve periodo, non riceverà dati che si discostano molto da quelli con cui è stato allenato. Questo poiché, come descritto in precedenza, i costi per nuovi stampi sono molto elevati e di conseguenza spesso si produce una medesima tipologia di prodotto senza eccessive variazioni (ad esempio modificando le caratteristiche del materiale plastico utilizzato).

Il secondo test, eseguito con il test set dei dati nuovi, a fronte di una buona accuracy vicina all'80%, presenta risultati di precision e recall estremamente più bassi rispetto al precedente test con i dati originali. Questo avviene poiché nei dati nuovi, raccolti nei mesi di novembre e dicembre 2020, è presente un elevato numero di falsi guasti, di conseguenza non sono presenti guasti reali. Oltre al fatto che, come confermato dal referente dell'azienda cliente STS Tecnopolimeri, in tale periodo di tempo non sono stati riscontrati guasti veri. Molte produzioni infatti erano già ampiamente conosciute, quindi non si sono prodotti articoli difettosi e non sono avvenuti guasti dovuti a errate configurazioni o materiali sbagliati. Si ricorda che, come descritto nei capitoli precedenti, i falsi guasti sono dovuti ad una errata configurazione del macchinario industriale che etichetta i fermi macchina come se fossero dei guasti. Di conseguenza il modello non li riconosce come guasti, in quanto effettivamente non lo sono. Tuttavia, tali problemi di registrazione dei fermi macchina possono essere risolti in futuro con una correzione alla configurazione del macchinario.

Infine, il terzo test, eseguito con il test set dei dati mix, è stato considerato come il più rappresentativo dell'intero sistema di manutenzione predittiva. Questo poiché utilizza un insieme maggiore di campioni, composto sia dai dati originali che dai dati nuovi registrati dai sensori presenti nel macchinario industriale. Vengono considerati quindi i dati generati in un intervallo temporale più ampio. In questo test il numero dei falsi negativi rimane elevato come nel secondo test, anche se, come descritto precedentemente, si tratta in realtà di fermi macchina segnalati come FN, che quindi in questo caso non sono guasti veri. Un modello per la manutenzione predittiva è importante che riduca il numero di FN, i quali non permettono di segnalare l'imminente guasto, causando danni al macchinario industriale. Invece i FP causano una inutile sospensione dell'attività del macchinario da parte dell'operatore per controllare lo stato della macchina, anche se non avverrà alcun guasto. Il risultato di accuracy del modello in questo test migliora, arrivando ad un buon 84%. La durata media delle produzioni nei dati considerati è di 2,5 ore, di conseguenza il modello di manutenzione predittiva che si è costruito dovrebbe essere in grado di predire mediamente con 2,5 ore di anticipo un guasto imminente al macchinario industriale. Questo poiché il modello è stato allenato per riconoscere la produzione

immediatamente precedente a quella durante la quale avverrà il guasto. Tale risultato, per essere più preciso, deve essere ulteriormente testato nella realtà per periodi di tempo più lunghi. Si ricorda che nel DataFrame 2 historical, sono etichettati come guasti tutti i campionamenti (che vengono registrati ogni secondo dai sensori) associati alle produzioni corrette immediatamente precedenti a quelle con guasto (settando la colonna is_breakdown a true). Questo in quanto i dati letti dai sensori cominciano in anticipo ad essere influenzati dall'imminente guasto che avverrà nella produzione successiva.

5.2 Deployment

In questa sesta ed ultima fase del modello CRISP-DM avviene il rilascio del sistema realizzato. Un modello di apprendimento non è particolarmente utile a meno che il cliente non possa accedere ai risultati prodotti. In base alle necessità e agli obiettivi del progetto la complessità del Deployment può variare notevolmente. Questa fase è composta dai seguenti task:

- **Pianificazione della distribuzione:** sviluppare e documentare un piano per la distribuzione del modello.
- **Pianificazione di monitoraggio e manutenzione:** sviluppare un monitoraggio completo e un piano di manutenzione costante per evitare problemi durante la fase operativa del modello costruito. Spesso ne è richiesta una messa a punto occasionale.
- **Produzione del report finale:** il team di progetto documenta un riepilogo del progetto che può includere una presentazione finale dei risultati raggiunti.
- **Revisione del progetto:** si conduce una retrospettiva del progetto, valutando cosa è andato bene, cosa ha presentato problematiche e come migliorare in futuro.

Durante quest'ultima fase di Deployment si è costruito il web server sul quale viene eseguito il modello allenato, oltre alla relativa pagina di configurazione. Successivamente, si è effettuato l'*upload* del web server sulla piattaforma di cloud computing Heroku, oltre a produrre un'analisi su come integrare il sistema di manutenzione predittiva all'interno

di Opera MES. Infine, si è progettata una variante di tale sistema che possa essere eseguita secondo un approccio di edge computing, analizzandone pregi e difetti.

5.2.1 Costruzione del web server e della pagina di configurazione

Durante la fase di Deployment si è innanzitutto costruito il web server sul quale verrà eseguito il modello di Machine Learning allenato in precedenza. Tale web server è stato implementato utilizzando il framework Flask, che permette di costruire applicazioni web, eseguendolo in un *virtual environment* di Python. Oltre al web server si è creata una pagina di configurazione, necessaria per impostare alcuni parametri del sistema di manutenzione predittiva. Come mostrato nella figura 5.7, la pagina di configurazione permette all'utente di scegliere il numero di sensori utilizzati, eseguire l'upload del file del modello allenato (con estensione *.sav*) e l'upload del file dello scaler (con estensione *.bin*) necessario per la standardizzazione dei nuovi dati. La pagina di configurazione è stata implementata con i linguaggi HTML e CSS, oltre ad utilizzare il framework Bootstrap per la costruzione dell'interfaccia grafica. I dati e i file inseriti nella pagina di configurazione vengono inviati al web server in esecuzione, tramite apposite chiamate HTTP POST. Il server riceve tali informazioni che gestisce con apposite funzioni, caricando e salvando i file del modello e dello scaler, oltre a creare i tre array di dimensione fissa per ogni sensore relativi agli storici breve, medio e lungo. Questi tre array memorizzano rispettivamente gli ultimi 10, 30 e 60 campionamenti, esattamente come accadeva per la costruzione del DataFrame 2 historical. Il sistema è gestito in modo dinamico tramite dizionari, per permettere di riadattarsi in base al numero di sensori che si scelgono di utilizzare. Inoltre è presente una funzione di test che riempie gli array utilizzando un dataset di esempio con i dati originali. Infine, nel web server è definita la funzione principale chiamata *get_data()*. Tale funzione viene eseguita quando il server riceve i nuovi dati tramite una chiamata HTTP POST, con i quali deve eseguire la predizione, rappresenta quindi l'omonimo *endpoint* che il client può interrogare. All'interno di *get_data()* vengono recuperati i dati contenuti nel body della chiamata HTTP POST e si memorizzano in un apposito dizionario. Successivamente viene costruito l'array che rappresenta il nuovo record del quale eseguire la predizione. Per ogni sensore si aggiunge il nuovo dato agli array dei relativi tre storici e si calcolano le rispettive medie e deviazioni standard

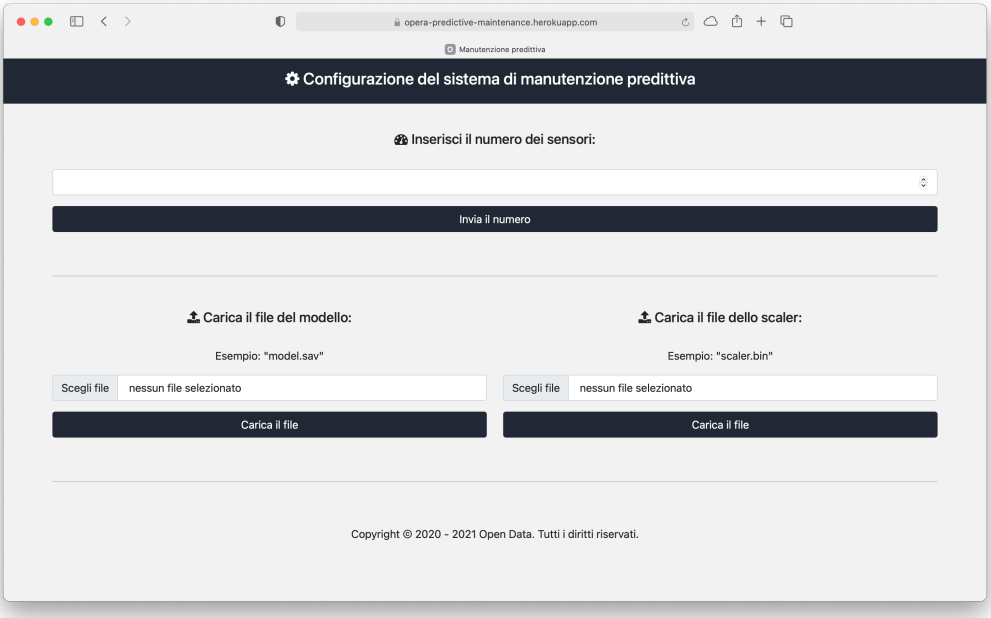

Figura 5.7: Screenshot della pagina di configurazione

sul breve, medio e lungo periodo. In questo modo, come durante la costruzione del DataFrame 2 historical, ad ogni sensore saranno associati 6 valori, che rappresentano le 6 colonne nel DataFrame 2 historical. Il nuovo record viene standardizzato utilizzando il file *scaler.bin* caricato tramite la pagina di configurazione e viene dato in input al modello allenato, anch'esso precedentemente caricato in fase di configurazione del sistema. Il modello esegue la predizione, decidendo se il nuovo record appartiene ad una produzione corretta o ad una produzione con guasto. Il risultato della predizione viene inviato come risposta dal server al client, indicando con il valore -1 il caso di produzione corretta e con il valore 1 il caso di produzione con guasto. Come mostrato nella figura 5.8, l'intero sistema appena descritto è stato testato inizialmente in locale, utilizzando il software Postman per eseguire le chiamate HTTP POST all'endpoint attivo e in ascolto sulla macchina locale, raggiungibile al seguente URL: *http://localhost:5000/getdata*. Successivamente, si sono eseguiti l'upload e l'esecuzione del sistema sul servizio di cloud computing Heroku, come descritto nel prossimo capitolo.

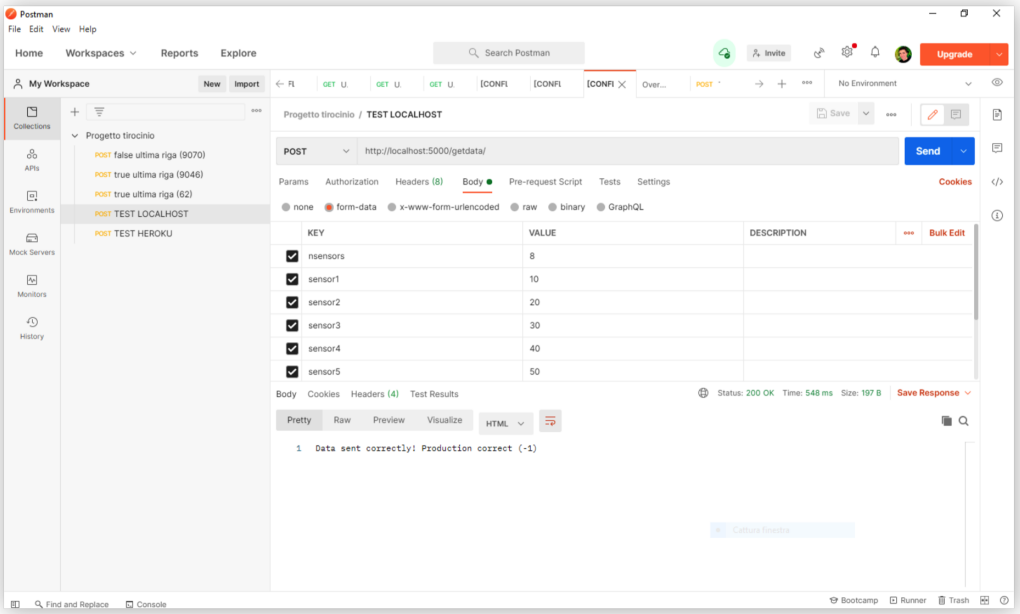

Figura 5.8: Screenshot di Postman durante il test in locale

5.2.2 Architettura del sistema con Cloud Computing

Come descritto nel precedente capitolo, dopo aver implementato il web server con Flask, realizzato la pagina di configurazione e testato il sistema in locale, si è eseguito il caricamento del progetto sulla piattaforma Heroku. È stata creata una nuova applicazione nel servizio di cloud computing, nella quale si è eseguito l'upload del progetto utilizzando i comandi del software di versionamento Git direttamente dal terminale, come mostrato nella figura 5.9 . In questo modo è stato possibile mantenere attiva l'esecuzione del sistema su un servizio di cloud computing, che fosse sempre raggiungibile e stabile per essere utilizzato in fase di Deployment. Di seguito si descrive l'architettura generale e il funzionamento dell'intero sistema di manutenzione predittiva utilizzando l'approccio che sfrutta il cloud computing. Nei prossimi capitoli si descriveranno in modo più approfondito alcune componenti del sistema, oltre alla proposta di un approccio alternativo che utilizza l'edge computing. Come mostrato nella figura 5.10, che rappresenta il primo schema progettuale, i dati grezzi vengono letti dai sensori posizionati nel macchinario

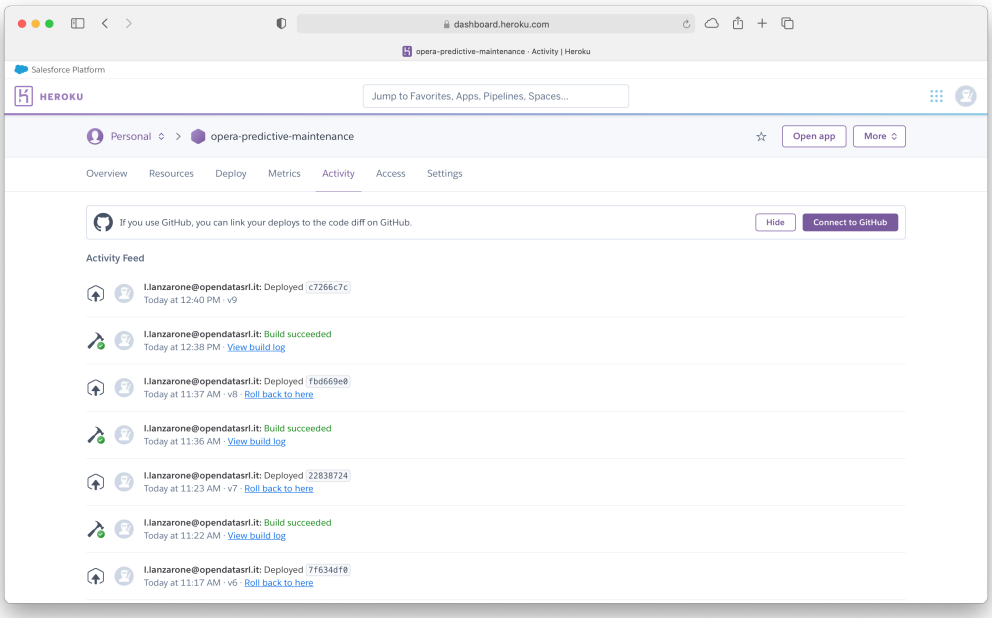

Figura 5.9: Screenshot della pagina di gestione del progetto di Heroku

industriale per lo stampaggio plastico a iniezione. Il PLC ha il compito di gestire il recupero di tali dati, si tratta di un computer integrato nel macchinario e specializzato nella gestione e nel controllo del processo produttivo dello stesso. Successivamente, attraverso il protocollo di comunicazione Euromap, il PLC comunica con il server di Opera MES installato nella rete locale dell'azienda cliente. In questo modo tutti i dati rilevati dai sensori vengono inviati ad Opera MES ad ogni determinato istante di tempo, in particolare avviene una lettura ogni secondo. Opera MES si occupa di registrare costantemente tutti i dati ricevuti nel database Microsoft SQL Server, nel quale è memorizzato lo storico completo di ogni attività che avviene nel sistema. Successivamente, come si descriverà in modo più approfondito nel prossimo capitolo, Opera MES invierà con un'apposita chiamata HTTP POST i dati letti dai sensori al servizio di cloud computing Heroku, sul quale è in esecuzione il web server realizzato con Flask. Quest'ultimo, come è stato descritto in modo più approfondito nel precedente capitolo, eseguirà una serie di operazioni. In particolare riceverà i nuovi dati, li elaborerà utilizzando i tre array per ogni

sensore che mantiene memorizzati sul breve, medio e lungo periodo, standardizzerà tali dati e li passerà in input al modello di Machine Learning, il quale restituirà in output la predizione. Tale output verrà inviato come risposta ad Opera MES, che si occuperà di mostrare i risultati nel client, accessibile all'operatore. Come si descriverà in modo più approfondito nei prossimi capitoli, su quest'ultimo verrà visualizzata la dashboard, nella quale l'operatore potrà osservare il comportamento del macchinario industriale in tempo reale, tutti i dati mostrati verranno recuperati dal database.

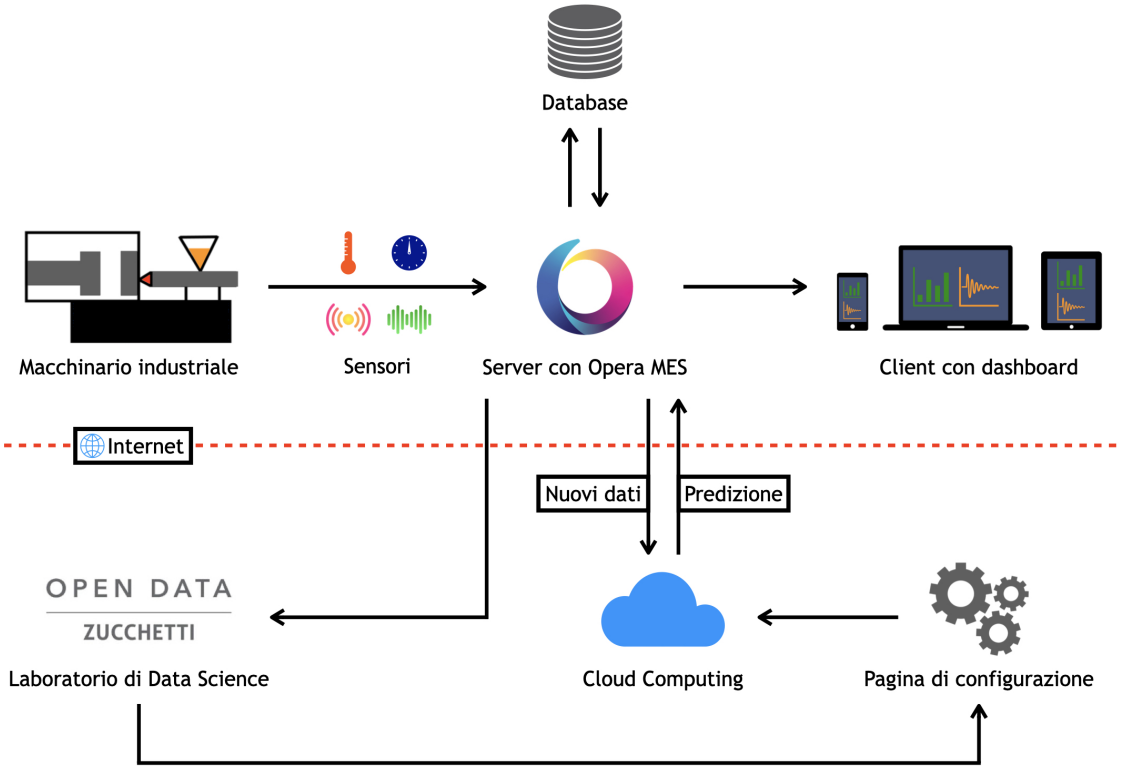

Figura 5.10: Schema dell'architettura del sistema con cloud computing

Il web server in esecuzione nel servizio di cloud computing Heroku, quindi, eseguirà ogni nuova richiesta utilizzando il modello di Machine Learning già allenato. Tale modello potrà essere aggiornato in base alle necessità, ad esempio nel caso in cui il sistema dovesse produrre predizioni errate, nel caso in cui si dovesse cambiare la configurazione

del macchinario industriale o ancora se l'articolo in plastica da stampare dovesse subire modifiche di produzione. In questi casi, il *laboratorio di Data Science*, cioè un team dedicato interno ad Open Data, si occuperà di recuperare da Opera MES lo storico dei dati, analizzarli e costruire un nuovo modello. Oppure eseguire un nuovo training su dati elaborati in modo diverso. Tale processo è sostanzialmente quello descritto nei capitoli precedenti, percorrendo le varie fasi del modello CRISP-DM, nel quale a partire dai dati grezzi si costruisce il modello di Machine Learning. Successivamente, il modello aggiornato, insieme al file degli scaler per la standardizzazione dei dati, dovranno essere caricati sul cloud di Heroku tramite l'apposita pagina di configurazione realizzata e descritta nel precedente capitolo. Tale pagina web è raggiungibile al seguente URL: *https://opera-predictive-maintenance.herokuapp.com*. Come mostrato nella figura 5.11, per testare l'esecuzione del sistema sul cloud computing si è utilizzato nuovamente il software Postman, con il quale si sono inviate le chiamate HTTP POST contenenti i dati letti dai sensori. In questo modo si è simulato il funzionamento di Opera MES.

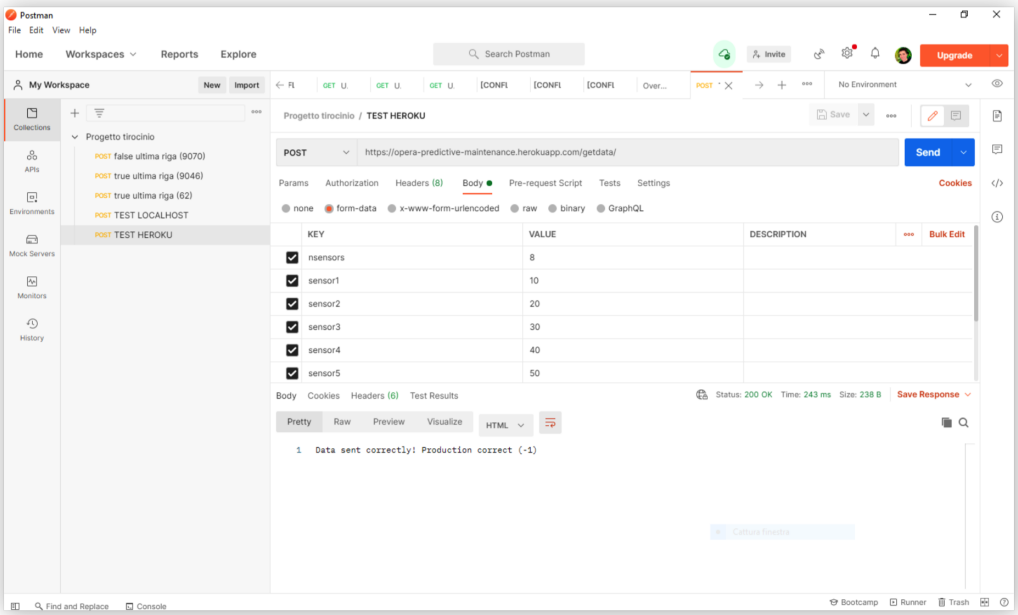

Figura 5.11: Screenshot di Postman durante il test con Heroku

5.2.3 Integrazione del sistema in Opera MES

L'intero sistema di manutenzione predittiva descritto nel presente elaborato è predisposto per essere integrato e utilizzato all'interno di Opera MES. L'integrazione in quest'ultimo può avvenire implementando un apposito *workflow* con JavaScript, il quale, ogni determinato intervallo di tempo si occuperà di inviare i nuovi dati ricevuti in tempo reale dai sensori del macchinario al sistema di manutenzione predittiva in esecuzione su Heroku. La comunicazione da parte di Opera MES avviene utilizzando una apposita chiamata HTTP POST che invierà i dati dei campionamenti nel body della richiesta all'endpoint attivo e in ascolto sul sistema di cloud computing, raggiungibile al seguente URL: *https://opera-predictive-maintenance.herokuapp.com/getdata/*.

Il sistema di manutenzione predittiva in esecuzione su Heroku potrà essere configurato dalla pagina di gestione dello stesso. Tale sistema riceverà i dati e li elaborerà come descritto nei capitoli precedenti. Successivamente, invierà la risposta con il risultato della predizione ad Opera MES, che si occuperà di salvare i dati nel database Microsoft SQL Server e di mostrare i risultati nel client accessibile all'operatore. La risposta avrà valore 1 nel caso in cui il risultato predetto sia relativo ad una produzione con guasto, mentre avrà valore -1 nel caso in cui il valore predetto sia relativo ad una produzione corretta. Poiché le produzioni eseguite nelle tipologie di macchinari industriali considerati hanno una durata media di 2,5 ore e il modello di Machine Learning è stato allenato per individuare le caratteristiche delle produzioni che sono immediatamente precedenti a quelle con guasto, si ottiene mediamente una predizione di eventuali guasti futuri con questo anticipo temporale.

Infine, quando il client che esegue Opera MES riceve la risposta dal server, la gestisce combinandola con i risultati ottenuti in precedenza dalle richieste inviate dal server ad Heroku. Tali risultati verranno mostrati all'operatore che potrà essere avvisato in tempo reale nel caso in cui sia previsto un guasto imminente al macchinario industriale. Nel front-end del client verrà mostrata una dashboard come quella ideata nei mockup realizzati con il software Balsamiq, tale interfaccia grafica è descritta nel prossimo capitolo.

5.2.3.1 Mockup della dashboard sul client

L'interfaccia grafica realizzata con il software Balsamiq e progettata per i client che eseguono Opera MES, mostra all'operatore che lavora a stretto contatto con il macchinario industriale una apposita dashboard. Quest'ultima può essere implementata con HTML, CSS, Bootstrap e apposite librerie JavaScript, in grado di costruire e gestire grafici che si aggiornano in modo dinamico. Come si può vedere nelle figure 5.12, 5.13, e 5.14, che mostrano i tre diversi stati di allerta del sistema, nella parte superiore della schermata sono presenti tre box distinti:

- **Prediction:** il primo box sulla sinistra, chiamato *Prediction*, mostra sempre all'operatore lo stato attuale del macchinario industriale, indicando se la situazione è normale (bassa presenza di possibili guasti rilevati, mostrato nella figura 5.12), moderata (media presenza di possibili guasti rilevati, mostrato nella figura 5.13) o critica (alta presenza di possibili guasti rilevati, mostrato nella figura 5.14). Per ottenere una risposta più precisa da mostrare all'operatore, anziché basarsi su singole predizioni, i livelli appena descritti saranno calcolati sui risultati dei tre intervalli temporali del secondo box. Potranno inoltre essere impostati in base a soglie scelte a seconda del tipo di macchinario industriale utilizzato e al livello di sensibilità desiderato dal cliente. Infine, in basso al primo box, è riportato il dato di accuracy del modello di predizione più aggiornato.

- **Historical View:** il secondo box centrale, chiamato *Historical View*, mostra un grafico a barre su tre diversi periodi: breve, medio e lungo. Come descritto precedentemente, tali intervalli temporali potranno essere scelti in fase di configurazione del sistema. Ogni barra del grafico mostrerà visivamente la percentuale di campionamenti classificati come guasti o come corretti contenuti nello specifico intervallo temporale e ottenuti come risposta dal modello in esecuzione su Heroku.

- **Live View:** il terzo box sulla destra, chiamato *Live View*, mostra il grafico a linee relativo all'andamento della predizione istante dopo istante, con l'indicazione della tipologia dell'ultima predizione eseguita.

Inoltre, nella parte inferiore della schermata, sono presenti una serie di grafici a linee per ogni sensore, che mostrano l'andamento dei valori letti dagli stessi e inviati al sistema

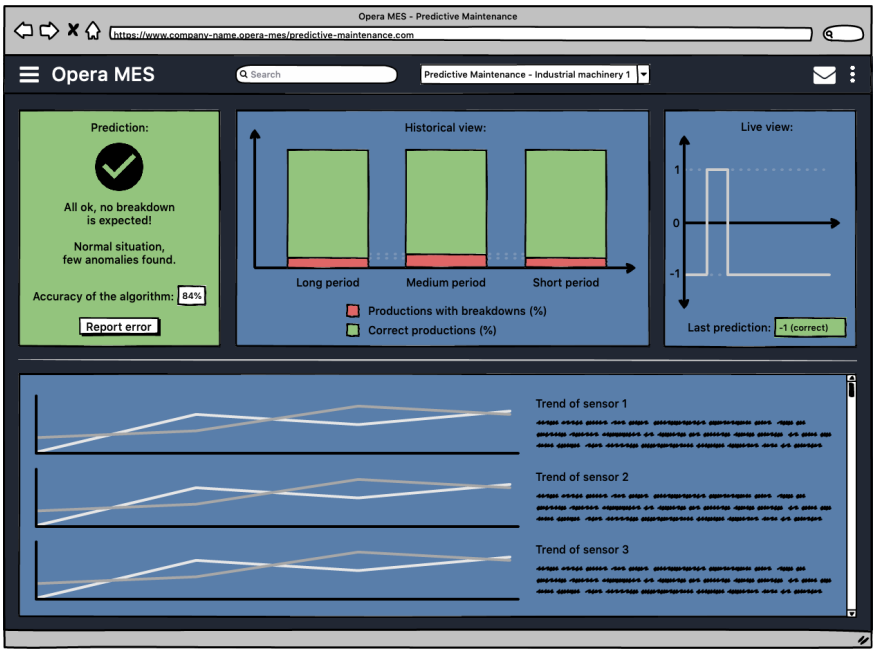

Figura 5.12: Mockup della dashboard sul client in situazione normale

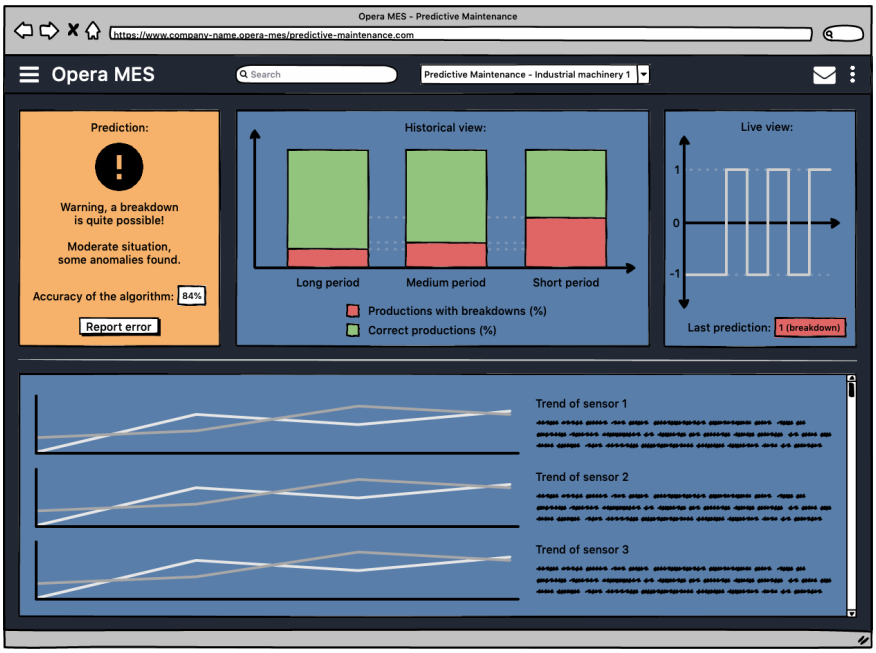

Figura 5.13: Mockup della dashboard sul client in situazione moderata

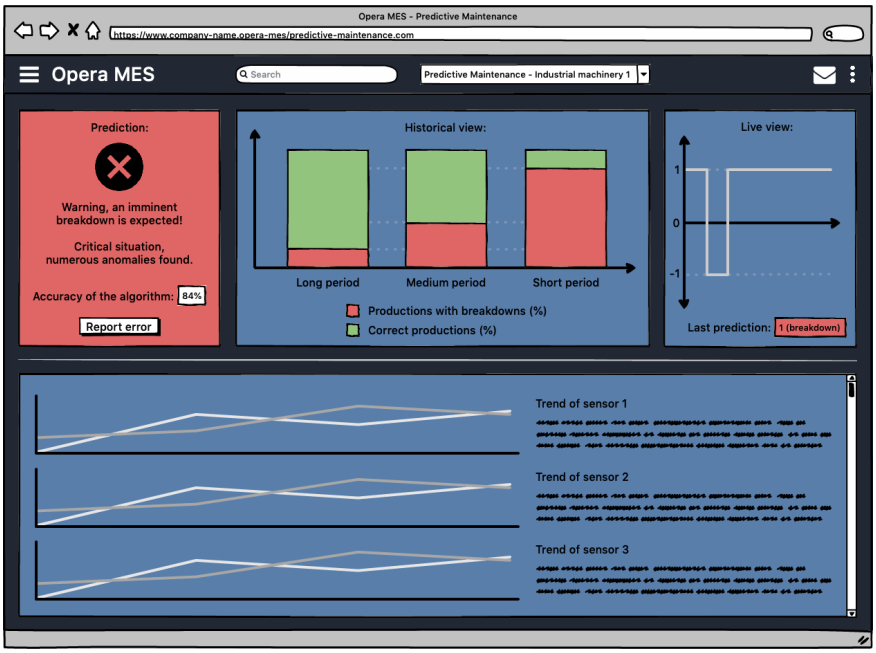

Figura 5.14: Mockup della dashboard sul client in situazione critica

di manutenzione predittiva. Tutti i grafici si aggiornano in modo dinamico per mostrare costantemente l'evoluzione del sistema, in modo che l'operatore possa avere la panoramica di quello che sta accadendo. Poiché il sistema al momento non è in grado di segnalare esplicitamente la tipologia di guasto imminente, in base alla sua esperienza e ai grafici che mostrano l'andamento dei campionamenti dei sensori, l'operatore potrà avere un'indicazione della tipologia di guasto che sta per verificarsi. In questo modo potrà agire direttamente sulle componenti del macchinario coinvolte nell'anomalia e controllarne lo stato. Infine, potrà essere implementata una funzionalità per permettere all'operatore di confermare se il guasto predetto dal modello di Machine Learning è realmente avvenuto o al contrario se è avvenuto un guasto che non è stato correttamente predetto. In questo modo potranno essere etichettati i dati che hanno prodotto predizioni errate ed essere analizzati successivamente per migliorare l'intero sistema.

5.2.4 Architettura del sistema con Edge Computing

La sempre maggiore diffusione dei dispositivi IoT e l'utilizzo dei servizi di cloud computing hanno favorito la nascita di un nuovo paradigma di elaborazione chiamato edge computing. Tale paradigma richiede che l'elaborazione dei dati venga eseguita ai margini della rete, in prossimità della fonte dei dati stessi. IDC, importante società di analisi di mercato, definisce così l'edge computing [73]:

> *"Una rete mesh di micro data center, in grado di elaborare e memorizzare dati critici localmente, e di trasmettere tutti i dati ricevuti e/o elaborati a un data center centrale o a un repository di cloud storage."*

Sostanzialmente tale topologia di rete, anche utilizzando sensori e sistemi elettronici di dimensioni contenute e costo ridotto, porta i componenti base di elaborazione, archiviazione e networking più vicino alle fonti che generano i dati. In questo modo l'edge computing offre notevoli vantaggi, come il miglioramento della latenza e quindi la riduzione del tempo di risposta del sistema, il risparmio sui costi di larghezza di banda necessari nell'utilizzo del cloud computing, un risparmio energetico sulle componenti wireless dei sensori, nonché vantaggi inerenti la sicurezza dei dati e la privacy.

Diverse applicazioni IoT necessitano di tempi di risposta molto brevi, potrebbero utilizzare dati sensibili, o ancora produrre una grande quantità di dati che possono rappresentare un carico eccessivo per la rete. Per questi motivi il cloud computing potrebbe non essere abbastanza efficiente o in grado di offrire un canale stabile di comunicazione per supportare applicazioni di questo genere. Nel progetto di manutenzione predittiva descritto fino ad ora, si utilizzano dati che sono particolarmente *time-sensitive*, cioè devono essere acquisiti ed elaborati in tempo reale per permettere all'operatore che sta lavorando con il macchinario industriale di avere la situazione dell'intero sistema sotto controllo in ogni istante di tempo. Come accennato, poiché i dati sono prodotti in misura sempre maggiore ai margini della rete, risulta più efficiente eseguirne l'elaborazione il più vicino possibile al luogo nel quale sono prodotti. Di conseguenza, nel progetto sviluppato si potrebbe spostare la computazione vicino al macchinario industriale del quale si vuole eseguire la manutenzione predittiva, piuttosto che utilizzare la computazione sul cloud.

Un'altra soluzione è quella di utilizzare i due differenti approcci in modo combinato, come verrà descritto successivamente nello schema progettuale [74].

Spostare tutte le attività di elaborazione sul cloud si è dimostrato essere un'alternativa valida per la computazione di grosse quantità di dati, poiché la potenza del cloud computing surclassa la capacità di calcolo che è in grado di offrire l'edge computing. Tuttavia, rispetto alla velocità di elaborazione dei dati, spesso la larghezza di banda della rete non è tale da permettere un collegamento sufficientemente veloce con il servizio cloud che si utilizza. Con la crescente quantità di dati generati dai dispositivi, la velocità del trasporto di tali dati sta diventando un cosiddetto *collo di bottiglia* per il paradigma del cloud computing, anche se tale situazione potrà migliorare nel tempo con l'utilizzo di reti sempre più veloci come la *fibra ottica* o il *5G*. Ad esempio, ogni secondo un veicolo a guida autonoma genera almeno 1 GB di dati e richiede un'elaborazione in tempo reale affinché l'automobile possa prendere la decisione corretta [75]. La larghezza di banda però non è sufficiente per la trasmissione di una tale quantità di dati e di conseguenza è necessario adottare un approccio di edge computing piuttosto che di cloud computing. Se tutti i dati devono essere inviati al cloud per l'elaborazione, il tempo di risposta diventerebbe troppo lungo, inoltre la larghezza di banda e l'affidabilità della rete verrebbero messe a dura prova per la loro capacità di supportare un elevato numero di altri veicoli a guida autonoma nella stessa area geografica. In questo caso, i dati devono essere elaborati direttamente sull'edge device per ottenere tempi di risposta più brevi, maggiore efficienza e minore pressione sull'intera infrastruttura di rete.

Nei sistemi di cloud computing, i dispositivi terminali che si trovano al margine della rete sono solitamente i cosiddetti consumatori di dati, ad esempio si può pensare ad un utente che guarda un video di YouTube sul proprio smartphone. Tuttavia, sempre più spesso gli utenti producono nuovi dati dai propri dispositivi, si assiste quindi ad un cambiamento fondamentale, l'utente passa dall'essere un semplice consumatore di dati ad esserne un produttore, oltre che un consumatore. Proprio questo nuovo approccio necessita di spostare sempre di più la computazione direttamente nel dispositivo edge. Ad esempio, questo vale per le applicazioni di intelligenza artificiale che sempre più spesso

sono eseguite direttamente sul dispositivo dell'utente. L'utilizzo di funzionalità come la realtà virtuale, la realtà aumentata o l'elaborazione del linguaggio naturale necessita di ottenere una risposta in tempo reale, oltre a dover mantenere la privacy e la sicurezza dei dati utilizzati senza divulgarli a terze parti. Per questi motivi è necessario che la computazione avvenga in prossimità della fonte dei dati, direttamente nel dispositivo edge. O almeno parte della computazione, per riuscire ad inviare una quantità inferiore di dati al cloud, nel caso in cui siano necessarie elaborazioni più complesse che non sono sostenibili direttamente dal dispositivo edge. Quest'ultimo può quindi elaborare i dati, archiviarli, mantenerli in una memoria cache, oltre a gestire eventuali richieste al servizio cloud per ulteriore potenza di calcolo. È necessario che l'intero sistema venga adeguatamente progettato in base al caso di utilizzo dello stesso, rispettando determinati vincoli e funzionalità richieste.

Il progetto di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione descritto nei capitoli precedenti è stato pensato e implementato per utilizzare il cloud computing. Con la seguente analisi implementativa, si vuole descrivere come è stato riprogettato l'intero sistema per utilizzare l'edge computing. La scelta nell'utilizzo di uno dei due approcci dovrà essere svolta insieme all'azienda cliente e alle sue necessità, ad esempio i tempi di risposta richiesti e il livello di privacy dei dati desiderato. Come descritto nel secondo schema progettuale, mostrato nella figura 5.15, l'architettura del sistema che utilizza l'edge computing è simile a quella descritta precedentemente per il cloud computing. La differenza sostanziale sta nel fatto che la predizione, intesa come l'esecuzione del modello sui nuovi dati ricevuti, avviene direttamente sul dispositivo edge. Tale dispositivo è posto in prossimità del macchinario industriale, quindi molto vicino alla fonte dalla quale vengono prodotti i dati. Il dispositivo edge comunica con Opera MES, ricevendo i nuovi dati e fornendo come risposta la predizione. Inoltre mantiene in memoria i tre array per ogni sensore sul breve, medio e lungo periodo, oltre a svolgere tutte le altre operazioni descritte in precedenza, poiché il server Flask viene eseguito totalmente in locale sul dispositivo edge. Come spiegato precedentemente per il sistema che utilizza il cloud computing, anche nel caso dell'edge computing il training del modello viene eseguito dal laboratorio di Data Science, che successivamente si occu-

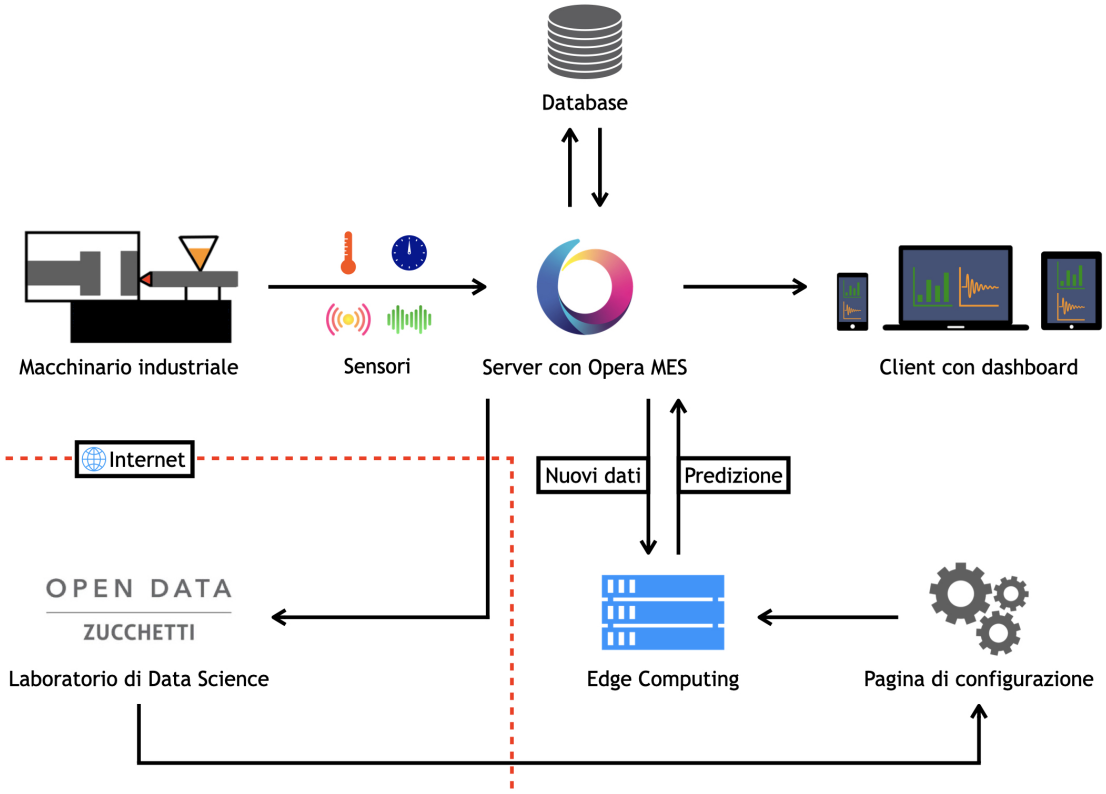

Figura 5.15: Schema dell'architettura del sistema con edge computing

però di caricare il modello allenato e il file degli scaler nell'edge device, collegandosi allo stesso tramite una connessione sicura VPN. L'azienda cliente, in base alle necessità e al contesto di utilizzo, potrà scegliere il modo in cui debba avvenire il training dei dati. Vi sono tre differenti opzioni:

- **Opzione 1:** training eseguito dal laboratorio di Data Science sul dispositivo edge installato in prossimità del macchinario industriale.
- **Opzione 2:** training eseguito dal laboratorio di Data Science su computer interni ad Open Data.
- **Opzione 3:** training eseguito dal laboratorio di Data Science utilizzando il cloud computing.

In accordo con il cliente si adotterà la soluzione migliore, la scelta avverrà in base alla privacy richiesta nel trattamento di tali dati e ai tempi di risposta desiderati. Ad esempio, se è richiesto un elevato livello di privacy è necessario che anche il training dei dati avvenga sul dispositivo edge, in questo caso tale dispositivo dovrà essere configurato in modo da avere la capacità di calcolo necessaria per eseguire tale operazione. La pagina di configurazione deve permettere al team del laboratorio di Data Science di eseguire in locale gli script per creare un nuovo DataFrame in un formato definito, utilizzando lo storico dei dati contenuti nel database. In tale pagina di configurazione sarà necessaria una sezione per eseguire l'upload del nuovo DataFrame esportato in formato CSV, quest'ultimo sarà il file di input per eseguire un nuovo training del modello direttamente sul dispositivo edge. Inoltre sarà necessaria una ulteriore sezione con le statistiche di utilizzo del dispositivo per permettere l'assistenza remota e controllarne le capacità di calcolo. La seconda alternativa consiste nell'eseguire il training con un livello di privacy ridotto poiché i dati vengono spostati ed elaborati sui computer interni ad Open Data. Infine, la terza opzione è quella che garantisce il minor livello di privacy ma potrebbe essere necessaria se la quantità dei dati è tale da dover necessariamente utilizzare il cloud computing per la maggiore potenza di calcolo che è in grado di offrire. In questo caso si potrebbe fare affidamento su servizi come *Heroku*, *Amazon Web Services* o *IBM Cloud*. Inoltre è possibile trovare un compromesso con il cliente, stipulando un apposito contratto nel quale si definiscono i dati necessari da utilizzare sul cloud computing, senza condividere quelli più sensibili e riservati, specificando anche per quanto tempo tali dati potranno essere utilizzati.

Il dispositivo edge installato in prossimità del macchinario industriale del quale si vuole implementare la manutenzione predittiva, dovrà possedere determinate caratteristiche tecniche in base al suo utilizzo. Come descritto precedentemente è necessario valutare se tale dispositivo dovrà occuparsi solo della comunicazione con Opera MES, dell'elaborazione dei dati, del mantenimento degli array storici e dell'esecuzione del modello, in questo caso non è necessaria una elevata potenza di calcolo. Oppure se tale dispositivo dovrà svolgere regolarmente anche il training del modello, in quest'ultimo caso la capacità computazionale dovrà essere maggiore. Altri vincoli legati alla scelta del dispositivo

sono il tempo massimo che il cliente è disposto ad attendere prima che venga allenato un nuovo modello, o ancora la necessità che il sistema di manutenzione predittiva continui a funzionare anche durante l'esecuzione del nuovo training del modello. Alcuni esempi di dispositivi edge che possono essere utilizzati sono quelli della serie *DragonBoard* di *Qualcomm*, i quali offrono basse e medie potenze di calcolo a prezzi contenuti [76]. O ancora i dispositivi della serie *Jetson* di *NVIDIA*, più potenti e dal costo più elevato, che possono essere utilizzati anche per eseguire il training dei dati [77]. Sul mercato sono disponibili comunque molteplici soluzioni alternative per ogni fascia di prezzo e di potenza di calcolo richiesta.

L'intero procedimento può essere ripetuto per utilizzare il sistema di manutenzione predittiva su più macchinari. Nel caso in cui nell'azienda cliente siano presenti più macchinari della stessa tipologia, si può utilizzare lo stesso dispositivo edge con lo stesso modello di Machine Learning in esecuzione, oppure modelli differenti se i macchinari sono di tipologia diversa. Nel caso in cui la potenza di calcolo richiesta risultasse eccessiva è preferibile utilizzare un dispositivo edge dedicato per ogni singolo macchinario industriale. Come descritto precedentemente, l'utilizzo dell'edge computing rispetto al cloud computing porta notevoli vantaggi, riassunti di seguito:

- Riduzione della latenza e quindi dei tempi di risposta del sistema.
- Notevole aumento della privacy e della sicurezza dei dati.
- Maggiore affidabilità del sistema.
- Risparmio della larghezza di banda necessaria.
- Risparmio sui costi di utilizzo del servizio di cloud computing.
- Risparmio energetico sulle comunicazioni non più necessarie, nel caso in cui si utilizzino sensori wireless.

Gli svantaggi principali nell'utilizzo dell'edge computing rispetto al cloud computing sono invece riassunti di seguito:

- Minore semplicità di configurazione, gestione e manutenzione da parte di Open Data, spesso i clienti non possiedono le competenze sistemistiche adeguate.
- Necessità di un costante monitoraggio del funzionamento del dispositivo edge.
- Costo iniziale di acquisto dei dispositivi edge da collegare ad ogni macchinario industriale.

In conclusione, la scelta di quale approccio utilizzare tra edge computing e cloud computing va eseguita in accordo con il cliente, in base alle sue esigenze e al contesto di utilizzo dell'intero sistema di manutenzione predittiva.

5.3 Problemi e soluzioni

In questo capitolo si sono riportati gli errori e le problematiche riscontrate durante la progettazione, l'implementazione e la validazione del sistema di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione. Per ogni problema sono specificate descrizione, possibili cause ed eventuali soluzioni che sono state applicate o che potrebbero essere applicate successivamente.

- Una delle problematiche riguarda la consistente presenza di guasti non causalizzati tra i dati relativi alle produzioni con guasto. Come accennato nei capitoli precedenti, si tratta di guasti registrati da Opera MES dei quali non è stata fornita una specifica causa dall'operatore che lavora a stretto contatto con il macchinario industriale. Per questi guasti, quindi, non è definita la relativa tipologia, di conseguenza si è tentato di automatizzare il processo di riconoscimento della causa del guasto utilizzando alcune tecniche di clustering, come K-Means, Agglomerative Clustering e DBSCAN. L'obiettivo è stato quello di associare ogni cluster rilevato ad una particolare causa, in modo da fornire una predizione più precisa all'operatore, indicando anche la tipologia di guasto, oltre all'avviso su un imminente guasto generico. In questo modo l'operatore avrebbe una prima indicazione del tipo di guasto che potrebbe presentarsi nel breve periodo, in modo da controllare direttamente le componenti del macchinario coinvolte. Come descritto nei capitoli

precedenti, la clusterizzazione non ha prodotto i risultati sperati. Di conseguenza non è stato possibile identificare la tipologia dei guasti a causa della complessità del dominio del problema, anche confrontandosi con il referente dell'azienda cliente STS Tecnopolimeri, che possiede maggiore esperienza nel funzionamento dei macchinari industriali analizzati, delle loro caratteristiche e dei dati generati dai relativi sensori. Per risolvere il problema è necessario che gli operatori, almeno per un primo periodo di tempo, indichino correttamente la tipologia di ogni guasto che avviene, in modo da creare un dataset con dati correttamente etichettati delle produzioni con guasto. Successivamente, tale dataset può essere utilizzato per allenare un modello di Machine Learning che utilizzi un approccio supervisionato in modo da automatizzare il riconoscimento della causa dei guasti. Un'alternativa può essere l'utilizzo di altre tecniche di Machine Learning che sfruttano un approccio di apprendimento non supervisionato, testando la loro efficacia rispetto a quelle già utilizzate in precedenza. Per arginare il problema, nella dashboard del client visualizzata dagli operatori, sono presenti una serie di grafici che si aggiornano in modo dinamico nella parte inferiore della schermata. Tali grafici si riferiscono all'andamento dei campionamenti effettuati dai sensori istante per istante. In questo modo, il sistema avvisa l'operatore di un imminente guasto e quest'ultimo può visionare velocemente i grafici dei sensori per avere un'indicazione della tipologia di guasto, basandosi anche sulla propria esperienza.

- Altra problematica è relativa sempre alle produzioni con guasto e riguarda la consistente presenza di dati etichettati come guasti che in realtà risultano essere dei fermi macchina. Come descritto nei capitoli precedenti, i fermi macchina sono situazioni in cui il macchinario industriale è attivo ma non sta producendo, non essendo impostato alcun programma da eseguire. Ad esempio, momenti in cui l'operatore è in pausa pranzo o vi sono cambi di turno tra gli operatori stessi. Tali situazioni sono registrate erroneamente come guasti in Opera MES. Anche in questo caso si è tentato di automatizzare il processo di riconoscimento dei fermi macchina, utilizzando le tecniche di clustering K-Means, Agglomerative Clustering e DBSCAN. L'obiettivo è stato quello di associare uno specifico cluster rilevato alla classe dei fermi macchina, per poterli rimuovere utilizzando un apposito filtro.

Uno dei possibili cluster riferiti ai fermi macchina era quello che raggruppava tutte le produzioni con durate estremamente elevate, compatibili con eventuali fermi macchina. Tuttavia, di tale ipotesi non si sono trovate prove sufficienti per essere considerata pienamente attendibile. Come appena descritto, poiché le tecniche di clusterizzazione non hanno prodotto i risultati sperati, anche confrontandosi con il referente esperto del dominio, non è stato possibile applicare un filtro per escludere i fermi macchina. Di conseguenza, tale rumore nei dati è rimasto anche nelle fasi successive di creazione dei DataFrame. Dai risultati descritti nel capitolo relativo alla fase di Evaluation, si nota infatti la forte presenza di fermi macchina classificati erroneamente come guasti falsi negativi. Per risolvere questa problematica è necessario un cambio alla configurazione del macchinario, in modo che invii ad Opera MES dati classificati correttamente come fermi macchina o come guasti.

- Lo script Python costruito per l'analisi della correlazione tra i dati registrati dai sensori presentava tempi di esecuzione particolarmente lunghi. Questo è dovuto alla grande quantità di dati da analizzare e all'elevato costo computazionale delle operazioni svolte, inclusa la costruzione dei grafici contenenti numerosi punti. Per tentare di velocizzare l'esecuzione dello script, si è utilizzato il *package multiprocessing* impostando il numero di *thread* da eseguire contemporaneamente. Inoltre si sono aumentate le risorse assegnate alla macchina virtuale e l'esecuzione dello script è avvenuta durante le ore notturne.

- Gli script Python realizzati per la costruzione dei DataFrame presentavano tempi di esecuzioni particolarmente lunghi, dovuti alle funzioni di ricerca delle produzioni tra loro vicine. Per risolvere il problema, si sono implementate funzioni ricorsive che minimizzano il costo computazionale e di conseguenza i tempi di esecuzione dello script. In particolare, ricercando le produzioni contenute all'interno di uno specifico intervallo di tempo, indicato dalla variabile *time_ delta*. Tale variabile aumenta progressivamente fino ad una soglia nel caso in cui non si trovino produzioni vicine. Successivamente, la ricerca della produzione più vicina ad un'altra avviene all'interno del sottoinsieme di produzioni recuperate precedentemente, anziché eseguire la ricerca sull'intero dataset.

- Durante l'upload del sistema di manutenzione predittiva sul servizio di cloud computing Heroku, eseguito utilizzando i comandi del software Git direttamente dal terminale, sono state riscontrate problematiche nel caricamento del progetto. In particolare, l'invio della prima *commit* è stato ripetutamente rifiutato da Heroku, in quanto alcune delle librerie incluse nel progetto non erano supportate dal servizio. Per risolvere il problema si è controllato quali fossero le librerie non necessarie all'esecuzione del sistema, escludendole dal caricamento sulla piattaforma di cloud computing.

Conclusioni ed estensioni future

Da quanto discusso in questa tesi si è compresa la sempre crescente importanza che ha l'evoluzione tecnologica nella società attuale, in particolare nel contesto dell'Industria 4.0 e in una delle sue applicazioni: la manutenzione predittiva. Si è compreso il processo che ha portato alla quarta rivoluzione industriale e i benefici che il settore produttivo può trarre da essa. Come nel caso della manutenzione predittiva, che permette un risparmio nei tempi, nei costi e nell'utilizzo di risorse, riuscendo ad anticipare un imminente guasto ad un macchinario industriale.

Si è compresa inoltre l'importanza delle tecniche di intelligenza artificiale, in particolare relative al Machine Learning e al Deep Learning, per realizzare un modello di apprendimento che permetta la manutenzione predittiva. Grazie alla Data Science è stato possibile comprendere e analizzare i fenomeni reali tramite l'utilizzo dei dati, che attraverso la loro analisi e manipolazione, abbinate alla conoscenza del dominio del problema, permettono molteplici possibilità di utilizzo.

Nel lavoro descritto nel presente elaborato è stato progettato, implementato e validato un sistema di manutenzione predittiva di macchinari industriali per lo stampaggio plastico a iniezione. Il progetto realizzato ha permesso una maggiore comprensione delle tematiche relative all'intelligenza artificiale e alla manutenzione predittiva, oltre all'utilizzo del processo CRISP-DM che ha accompagnato l'intero progetto. Il modello selezionato presenta un'accuracy dell'84% e una predizione media del guasto con 2,5 ore di anticipo. Sono state inoltre discusse le problematiche riscontrate e le soluzioni applicate. Il sistema proposto rappresenta una nuova funzionalità che, con opportune modifiche e miglioramenti,

potrà essere integrata all'interno di Opera MES, il software di Manufacturing Execution System prodotto da Open Data. Di seguito sono raccolte le possibili estensioni future relative al progetto realizzato:

- Un'estensione futura, come accennato precedentemente, riguarda la possibilità di integrare il sistema di manutenzione predittiva all'interno di Opera MES. Come spiegato nel presente elaborato, il sistema è stato predisposto per essere implementato come nuova funzionalità in Opera MES, oltre ad averne descritto l'architettura nei casi di utilizzo con il cloud computing e con l'edge computing.

- Relativamente all'esecuzione del sistema di manutenzione predittiva realizzato utilizzando un approccio con edge computing, un'ulteriore estensione futura riguarda il test dello stesso nel contesto reale. In modo tale da comprendere pregi e difetti dell'architettura proposta.

- Un'altra estensione futura riguarda la possibilità di allenare e testare il sistema con dati che siano più puliti, risolvendo i problemi descritti relativi ai guasti non causalizzati e ai fermi macchina. Il processo di raccolta dei dati dal macchinario può essere perfezionato, in modo tale da ottenere migliori risultati di accuratezza e precisione del modello.

- Un'altra estensione futura sempre inerente ai dati riguarda la possibilità di allenare e testare il sistema di manutenzione predittiva realizzato con una quantità maggiore di dati, anche di tipologia differente. Poiché tutti gli script Python sono stati progettati e implementati in modo dinamico, si potrebbero utilizzare facilmente i dati provenienti da più sensori o da sensori differenti.

- Un'ulteriore estensione futura riguarda la possibilità di applicare lo stesso approccio, utilizzato per la realizzazione del progetto, anche ad altre tipologie di macchinari industriali. Il sistema è stato tarato sui dati provenienti dai sensori di una pressa ad azionamento elettrico. Il modello potrebbe quindi essere nuovamente allenato per testare il sistema su presse ad azionamento idraulico o ibrido, o ancora ad altre categorie di macchinari industriali con le dovute modifiche.

- Un'ultima estensione futura riguarda la possibilità di ampliare il sistema di manutenzione predittiva includendo anche la cosiddetta qualità predittiva. Si potrebbero utilizzare i dati relativi ai difetti degli articoli prodotti che sono stati estratti inizialmente dal database, in modo da predire la qualità dell'articolo in plastica da realizzare oltre ai guasti del macchinario industriale.

In conclusione si è compreso che il valore dei dati e delle informazioni oggi è nettamente superiore a qualsiasi altra forma di contributo. L'Industria 4.0 e le sue applicazioni di intelligenza artificiale come la manutenzione predittiva, possono apportare notevoli benefici all'efficienza degli impianti produttivi, riducendo gli sprechi di materiali, di risorse e i costi di riparazione dei guasti ai macchinari industriali.

Elenco delle figure

Elenco delle tabelle

Bibliografia e sitografia

[1] Ron Davies, "Industry 4.0: Digitalisation for productivity and growth". Parlamento Europeo, 2015.
[https://www.europarl.europa.eu/thinktank/en/document.html?reference=EPRS_BRI%282015%29568337]

[2] Andreja Rojko, "Industry 4.0 Concept: Background and Overview". International Journal of Interactive Mobile Technologies, 2017.
[https://online-journals.org/index.php/i-jim/article/view/7072/0]

[3] Von Henning Kagermann Wolf-Dieter Lukas, "Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution". Ingenieur, 2011.
[https://www.ingenieur.de/technik/fachbereiche/produktion/industrie-40-mit-internet-dinge-weg-4-industriellen-revolution]

[4] Jacqueline Zonichenn Reis e Rodrigo Franco Gonçalves, "The Role of Internet of Services (IoS) on Industry 4.0 Through the Service Oriented Architecture (SOA)". International Conference Advances in Production Management Systems, 2018.
[https://hal.inria.fr/hal-02177890]

[5] Flaviano Celaschi, Loredana Di Lucchio e Lorenzo Imbesi, "Design e phigital production: progettare nell'era dell'Industria 4.0". MD Journal, 2017.
[http://materialdesign.it/media/formato2/allegati_6291.pdf]

[6] Area Industria e Innovazione, "Approfondimento sulle tecnologie abilitanti Industria 4.0". Assolombarda Confindustria Milano, Monza e Brianza, 2016.

[http://www.assolombarda.it/innovare-per-competere/approfondimento-sulle
-tecnologia-abilitanti-industria-4.0]

[7] Giambattista Gruosso, "Le tecnologie abilitanti dell'Industria 4.0". Fondirigenti, Ricomincio da 4, 2017.
[https://ricomincioda4.fondirigenti.it/le-tecnologie-abilitanti-dellindustria-4-0]

[8] Luca Beltrametti, Nino Guarnacci, Nicola Intini e Corrado La Forgia, "La fabbrica connessa. La manifattura italiana (attra)verso Industria 4.0". Guerini e Associati, 2017.
[https://www.amazon.it/fabbrica-connessa-manifattura-italiana-Industria-ebook/
dp/B072N6TT1B]

[9] Ministero dello Sviluppo Economico, "Piano nazionale Industria 4.0". Governo Italiano, 2017.
[https://www.mise.gov.it/images/stories/documenti/Piano_Industria_40.pdf]

[10] Markus Lorenz, Michael Rüßmann, Manuela Waldner, Pascal Engel, Michael Harnisch e Jan Justus, "Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries". Boston Consulting Group, 2015.
[https://www.bcg.com/it-it/publications/2015/engineered_products_project
_business_industry_4_future_productivity_growth_manufacturing_industries]

[11] "Manifattura additiva: la seconda tecnologia abilitante dell'Industria 4.0". Focus Industria 4.0, 2019.
[https://www.focusindustria40.com/manifattura-additiva]

[12] Rick Van Krevelen e Ronald Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations". International Journal of Virtual Reality, 2010.
[https://www.researchgate.net/publication/279867852_A_Survey_of_Augmented
_Reality_Technologies_Applications_and_Limitations]

[13] "Integrazione verticale e orizzontale: la quinta tecnologia abilitante dell'Industria 4.0". Focus Industria 4.0, 2020.
[https://www.focusindustria40.com/integrazione-verticale-e-orizzontale]

[14] "Cloud Computing: la settima tecnologia abilitante dell'Industria 4.0". Focus Industria 4.0, 2020.
[https://www.focusindustria40.com/cloud-computing]

[15] Suneel Wattal e Ajay Kumar, "Cloud computing - An emerging trend in information technology". International Conference on Issues and Challenges in Intelligent Computing Techniques, 2014.
[https://www.researchgate.net/publication/271546381_Cloud_computing_-_An_emerging_trend_in_information_technology]

[16] Yongxin Liao, Fernando Deschamps, Fernando Deschamps, Eduardo Rocha Loures e Luiz Felipe Ramos, "Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal". International Journal of Production Research, 2017.
[https://www.researchgate.net/publication/315670892_Past_present_and_future_of_Industry_40_-_a_systematic_literature_review_and_research_agenda_proposal]

[17] Ministero dello Sviluppo Economico, "Piano nazionale Transizione 4.0". Governo Italiano, 2020.
[https://www.mise.gov.it/index.php/it/transizione40]

[18] "The Future of Jobs Report 2020". World Economic Forum, 2020.
[http://www3.weforum.org/docs/WEF_Future_of_Jobs_2020.pdf]

[19] H. M. Hashemian, "State-of-the-Art Predictive Maintenance Techniques". IEEE, 2010.
[https://ieeexplore.ieee.org/document/5659691]

[20] Omkar Motaghare, Anju S Pillai e K.I. Ramachandran, "Predictive Maintenance Architecture". IEEE, 2018.
[https://ieeexplore.ieee.org/document/8782406]

[21] Cheng Cheng, Bei-ke Zhang e Dong Gao, "A Predictive Maintenance Solution for Bearing Production Line Based on Edge-Cloud Cooperation". IEEE, 2019.
[https://ieeexplore.ieee.org/document/8996482]

[22] Ana Cachada, Jose Barbosa, Paulo Leitño, Carla A.S. Gcraldcs, Leonel Deusdado, Jacinta Costa, Carlos Teixeira, João Teixeira, António H.J. Moreira, Pedro Miguel Moreira e Luís Romero, "Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture". IEEE, 2018.
[https://ieeexplore.ieee.org/document/8502489]

[23] Vincent F. A. Meyer zu Wickern, "Challenges and Reliability of Predictive Maintenance". Rhein-Waal University Of Applied Sciences, Faculty of Communication and Environment, 2019.
[https://www.researchgate.net/publication/331951459_Challenges_and_Reliability_of_Predictive_Maintenance]

[24] Deepak Jakhar Ishmeet Kaur, "Artificial intelligence, machine learning and deep learning: definitions and differences". Clinical and Experimental Dermatology, 2019.
[https://www.researchgate.net/publication/334014738_Artificial_intelligence_machine_learning_deep_learning_Definitions_and_differences]

[25] Yann LeCun, Yoshua Bengio e Geoffrey Hinton, "Deep learning". Nature, 2015.
[https://www.nature.com/articles/nature14539]

[26] Vasant Dhar, "Data Science and Prediction". Communications of the ACM, 2013.
[https://cacm.acm.org/magazines/2013/12/169933-data-science-and-prediction/fulltext]

[27] Chikio Hayashi, Noboru Ohsumi, Hans H. Bock, Keiji Yajima, Yutaka Tanaka e Y. Baba, "Data Science, Classification, and Related Methods". Springer, 1998.
[https://www.springer.com/it/book/9784431702085]

[28] Tony Hey, Anthony J. G. Hey, Stewart Tansley e Kristin Michele Tolle, "The Fourth Paradigm: Data-intensive Scientific Discovery". Microsoft Research, 2009.
[https://books.google.it/books?id=oGs_AQAAIAAJ&redir_esc=y]

[29] Sarah Guido e Andreas C. Mueller "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly Media, 2016.

[https://www.amazon.it/Introduction-Machine-Learning-Python-Sarah/dp/1449369413]

[30] Ornella Colpani, "Machine learning: the ability to predict applied to research and clinical practice". Giornale Italiano di Farmacoeconomia e Farmacoutilizzazione, 2019.
[http://www.sefap.it/web/upload/GIFF2019-4_5_11.pdf]

[31] Daniel R. Schrider e Andrew D. Kern, "Supervised Machine Learning for Population Genetics: A New Paradigm". Trends in Genetics, 2018.
[https://www.researchgate.net/publication/322397567_Supervised_Machine_Learning_for_Population_Genetics_A_New_Paradigm]

[32] Xiangying Wang e Yixin Zhong, "Statistical learning theory and state of the art in SVM". The Second IEEE International Conference on Cognitive Informatics, 2003.
[https://ieeexplore.ieee.org/abstract/document/1225953]

[33] Michael W. Berry, Azlinah Mohamed e Bee Wah Yap, "Supervised and Unsupervised Learning for Data Science". Springer International Publishing, 2019.
[https://books.google.it/books/about/Supervised_and_Unsupervised_Learning_for.html?id=0n6qzQEACAAJ&redir_esc=y]

[34] T. Soni Madhulatha, "An Overview on Clustering Methods". IOSR Journal of Engineering, 2012.
[https://arxiv.org/abs/1205.1117]

[35] Christopher M. Bishop, "Pattern Recognition And Machine Learning". Springer Nature, 2006.
[https://www.amazon.it/Pattern-Recognition-Machine-Learning-Christopher/dp/0387310738]

[36] Sonish Sivarajkumar, "ReLU - Most popular Activation Function for Deep Neural Networks". Medium, 2019.
[https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda]

[37] Yusuf Sani, Ahmed Mohamedou, Khalid Ali, Anahita Farjamfar, Mohamed Azman e Solahuddin Shamsuddin, "An Overview of Neural Networks Use in Anomaly Intrusion Detection Systems". IEEE, 2009.
[https://ieeexplore.ieee.org/document/5443289]

[38] Ian Goodfellow, Yoshua Bengio e Aaron Courville, "Deep Learning". MIT Press, 2015.
[https://books.google.it/books/about/Deep_Learning.html?id=Np9SDQAAQBAJ&redir_esc=y]

[39] Sebastian Ruder "An overview of gradient descent optimization algorithms". Cornell University, 2016.
[https://arxiv.org/abs/1609.04747]

[40] Saad Albawi, Tareq Abed Mohammed e Saad Al-Zawi, "Understanding of a convolutional neural network". IEEE, 2017.
[https://ieeexplore.ieee.org/abstract/document/8308186]

[41] Keiron O'Shea e Ryan Nash, "An Introduction to Convolutional Neural Networks". Cornell University, 2015.
[https://arxiv.org/abs/1511.08458]

[42] Guojun Wang, Weijun Li, Liping Zhang, Linjun Sun, Peng Chen, Lina Yu e Xin Ning, "Encoder-X: Solving Unknown Coefficients Automatically in Polynomial Fitting by Using an Autoencoder". IEEE, 2021.
[https://ieeexplore.ieee.org/document/9336312]

[43] Xuedan Du, Yinghao Cai, Shuo Wang e Leijie Zhang, "Overview of deep learning". IEEE, 2016.
[https://ieeexplore.ieee.org/abstract/document/7804882]

[44] R. Wirth e Jochen Hipp "CRISP-DM: Towards a Standard Process Model for Data Mining". Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining, 2000.

[https://www.researchgate.net/publication/239585378_CRISP-DM_Towards_a_standard_process_model_for_data_mining]

[45] Ana Azevedo e Manuel Filipe Santos, "KDD, SEMMA and CRISP-DM: a parallel overview". IADIS European Conference on Data Mining, 2008.
[https://www.researchgate.net/publication/220969845_KDD_semma_and_CRISP-DM_A_parallel_overview]

[46] Sito web "Open Data".
[https://www.opendatasrl.it]

[47] Benoît Saenz de Ugarte, Abdelhakim Artiba e Robert Pellerin, "Manufacturing execution system - A literature review". The Management of Operations, 2009.
[https://www.researchgate.net/publication/238181581_Manufacturing_execution_system_-_A_literature_review]

[48] Sito web "Opera MES".
[https://www.operames.it]

[49] Sito web "BMB".
[http://www.bmb-spa.com]

[50] Sito web "STS Tecnopolimeri".
[https://www.ststech.it]

[51] Giorgio Bertacchi, "Manuale dello stampaggio progettato". Tecniche Nuove, 2011.
[https://www.tecnichenuove.com/prodotto/manuale-dello-stampaggio-progettato]

[52] Hassan Ismail Fawaz, "Timeseries classification from scratch". Keras, 2020.
[https://keras.io/examples/timeseries/timeseries_classification_from_scratch]

[53] Zhiguang Wang, Weizhong Yan e Tim Oates, "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline". Cornell University, 2016.
[https://arxiv.org/abs/1611.06455]

[54] Pavithra Vijay, "Timeseries anomaly detection using an Autoencoder". Keras, 2020.
[https://keras.io/examples/timeseries/timeseries_anomaly_detection]

[55] Sito web "Microsoft SQL Server".
[https://www.microsoft.com/it-it/sql-server/sql-server-2019]

[56] Sito web "Python".
[https://www.python.org]

[57] Marco Buttu, "Programmare con Python. Guida completa". Edizioni LSWR, 2014.
[https://www.amazon.it/Programmare-Python-Guida-completa-Marco/dp/8868950243]

[58] Sito web "NumPy".
[https://www.numpy.org]

[59] Sito web "SciPy".
[https://www.scipy.org]

[60] Sito web "Matplotlib".
[https://www.matplotlib.org]

[61] Sito web "Seaborn".
[https://seaborn.pydata.org]

[62] Sito web "Pandas".
[https://pandas.pydata.org]

[63] Sito web "JupyterLab".
[https://www.jupyter.org]

[64] Sito web "Scikit-learn".
[https://scikit-learn.org]

[65] Sito web "Keras".
[https://keras.io]

[66] Sito web "Flask".
[https://palletsprojects.com/p/flask]

[67] Sito web "HTML".
[https://html.spec.whatwg.org]

[68] Sito web "CSS".
[https://www.w3.org/TR/CSS]

[69] Sito web "Bootstrap".
[https://getbootstrap.com]

[70] Sito web "Postman".
[https://www.postman.com]

[71] Sito web "Heroku".
[https://www.heroku.com]

[72] Sito web "Balsamiq".
[https://www.balsamiq.com]

[73] "Edge computing, cos'è, come funziona e come implementarlo". ZeroUno, 2020.
[https://www.zerounoweb.it/techtarget/searchdatacenter/edge-computing
-cose-come-implementarlo]

[74] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li e Lanyu Xu, "Edge Computing: Vision and Challenges". IEEE Internet of Things Journal, 2016
[https://ieeexplore.ieee.org/document/7488250]

[75] Lucas Mearian, "Self-driving cars could create 1GB of data a second". Computerworld, 2013.
[https://www.computerworld.com/article/2484219/self-driving-cars-could-create
-1gb-of-data-a-second.html]

[76] Sito web "Qualcomm DragonBoard".
[https://developer.qualcomm.com/hardware/dragonboard-410c]

[77] Sito web "NVIDIA Jetson".
[https://www.nvidia.com/it-it/autonomous-machines/embedded-systems]

Ringraziamenti

Vorrei spendere alcune righe di questa mia Tesi di Laurea Magistrale per ringraziare tutti coloro che mi hanno sostenuto e aiutato durante questo percorso universitario, un percorso impegnativo e ricco di soddisfazioni. Desidero innanzitutto ringraziare il Prof. Marco Di Felice, relatore di questa tesi, per il tempo che mi ha dedicato, la sua professionalità e competenza. Un ringraziamento particolare va ai miei correlatori, Dott. Gerardo Fabrizio e Dott. Davide Di Donato. Il primo per avermi permesso di svolgere il tirocinio per tesi presso la sua azienda, Open Data, e il secondo per avermi supportato e seguito nella realizzazione del progetto. Questo ringraziamento è rivolto anche agli altri colleghi, Dott. Federico Fabrizio e Dott. Leonardo Fabrizio. Desidero inoltre ringraziare il Dott. Matteo Pigliapoco, referente dell'azienda cliente STS Tecnopolimeri, e il Prof. Sergei Bezobrazov per aver contribuito in alcune scelte durante la realizzazione del progetto.

Il ringraziamento più grande va a tutta la mia famiglia, che mi ha sempre supportato moralmente ed economicamente in questo percorso, grazie infinite mamma e papà per tutto quello che fate per me. Desidero ringraziare mia sorella Sonia, alla quale voglio molto bene e che presto mi farà diventare zio di Camilla. Un ringraziamento di cuore va ai miei zii, Clara e Adelchi, da sempre disponibili e presenti nella mia vita. Grazie anche al nostro amato labrador Kira, che non c'è più, e al nostro amato golden retriever Derek, due cani molto speciali. Desidero inoltre ringraziare tutti i miei amici, grazie a tutti voi per i momenti di gioia trascorsi insieme. Ringrazio anche i miei colleghi e amici di corso, in particolare Cristian e Andrea, grazie davvero per esserci stati, rendendo più piacevoli questi anni di università. Infine un grazie a me stesso, per essere riuscito con tanto impegno a raggiungere questo importante traguardo, punto di arrivo e di inizio nella mia vita.

Lorenzo Biagio Lanzarone *Bologna, 18 marzo 2021*