Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Ingegneria e Scienza Informatiche

STUDIO E SPERIMENTAZIONE SU MANUTENZIONE PREDITTIVA IN AMBITO MANIFATTURIERO

Relatore:
Chiar.mo Prof.
Vittorio Maniezzo

Presentata da:
Luca Agatensi

Correlatori:
Chiar.mo Prof.
Antonella Carbonaro
Ing. ***Maurizio Pensato***

Sessione III
Anno Accademico 2018-2019

Introduzione

La Manutenzione Predittiva (Predictive Maintenance - *PdM*) è una pratica che viene utilizzata per ottimizzare i piani di manutenzione delle risorse, attraverso la previsione di guasti alle stesse, con tecniche che sfruttano la predizione di comportamenti sulla base di dati passati.

L'applicazione di *PdM* può portare numerosi vantaggi alle aziende, tra cui, la riduzione dei tempi di inattività e l'aumento della qualità del prodotto.

Le aziende più interessate alla *PdM* sono quelle che si occupano di produzione manifatturiera, che è l'area di interesse in questa tesi, ma anche tutte quelle che fanno della efficienza degli impianti una loro prerogativa.

Tuttavia, i metodi di *PdM* affrontano molteplici sfide, che possono limitarne o impedirne l'utilizzo. È importante raccogliere queste sfide per poter evidenziare l'affidabilità di questi metodi.

Le sfide tecnologiche sono quelle che più riguardano questa tesi di progetto che ha come obiettivo quello di sfruttare le moderne tecnologie di *Machine Learning* per fare manutenzione predittiva su un caso reale.

Infatti la tesi è stata realizzata in collaborazione con *SPOT Software Srl* e con *Cefla* che hanno messo a disposizione un problema e dei dati reali.

SPOT Software Srl e *Cefla* hanno fornito l'autorizzazione per essere nominati all'interno di questo documento.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Manutenzione Predittiva

1.1 Introduzione

Il degrado delle risorse e il calo di prestazioni è un problema di primo piano tra le aziende, soprattutto quando portano ad una indisponibilità delle risorse considerate essenziali o a drastici cali dell'efficienza[1].

La *Manutenzione Preventiva* è nata con lo scopo di prevenire guasti e tutto ciò che circonda le risorse sensibili, attraverso l'inserimento di cicli di manutenzione.

Di seguito il concetto di *Manutenzione Preventiva* si è evoluto in *Manutenzione Predittiva* (da qui in poi *PdM - Predictive Maintenance*) che mira a prevedere i guasti analizzando le risorse correlate tra loro e il loro comportamento passato per consentire la scelta di un momento ottimale per la manutenzione.

I metodi di *PdM* sono utilizzati in particolare nei dispositivi IoT[2], applicati nel campo dell'industria. Questa tesi si concentra sull'industria manifatturiera in quanto oggetto del caso di studio.

Nonostante il suo uso già frequente, la *PdM* è ancora un campo molto ricercato con molti paper, pubblicati negli ultimi anni, che ne fanno riferimento, ad es. [3] [4] [5].

Tuttavia, rimango alcune domande alle quali sarebbe necessario rispon-

dere: i metodi di *PdM* sono effettivamente affidabili? Quali vantaggi portano alle aziende che ne fanno uso? Quali sono le sfide tecnologiche che la *PdM* deve affrontare?

In questo primo capitolo si cercherà di rispondere a queste domande e dimostrare l'affidabilità della *PdM*. Tutto ciò può servire a supportare le aziende nella loro decisione in merito all'introduzione e all'applicazione della *PdM*. Inoltre, questa tesi può essere una base per approfondire la ricerca in questo ambito.

Per definire e confermare l'affidabilità della *PdM*, devono essere considerati e affrontati quattro argomenti:

- Concetti
- Goals
- Funzionalità
- Sfide Tecnologiche

Questo primo capitolo, per cui, si occuperà di mostrare ognuno dei punti presentati che riguardano la Manutenzione Predittiva[6].

1.2 Concetti di Base

Nelle aziende manifatturiere vengono utilizzate diverse strategie di manutenzione: *Manutenzione Reattiva*, *Manutenzione Preventiva* e *Manutenzione Predittiva*[5](la figura 1.1 mostra una ulteriore classificazione). La più tradizionale di queste strategie è la *Manutenzione Reattiva*, che ha inizio con la correzione del guasto solamente dopo che esso si è verificato. Al contrario, la *Manutenzione Preventiva* mira ad adottare misure preventive in anticipo in modo tale da prevenire i guasti.

Un difetto importante delle strategie di *Manutenzione Preventiva* è il fatto che non prevedano che la condizione corrente della risorsa influenzi il suo programma di manutenzione[7]. *CBM* ovvero *Condition Based Monitoring*

è una delle possibili implementazioni della *PdM* che, in contrasto, include le misurazioni delle condizioni delle risorse nella pianificazione della manutenzione. Ad esempio, le caratteristiche che sono monitorate più frequentemente sono la temperatura o la vibrazione di una macchina.

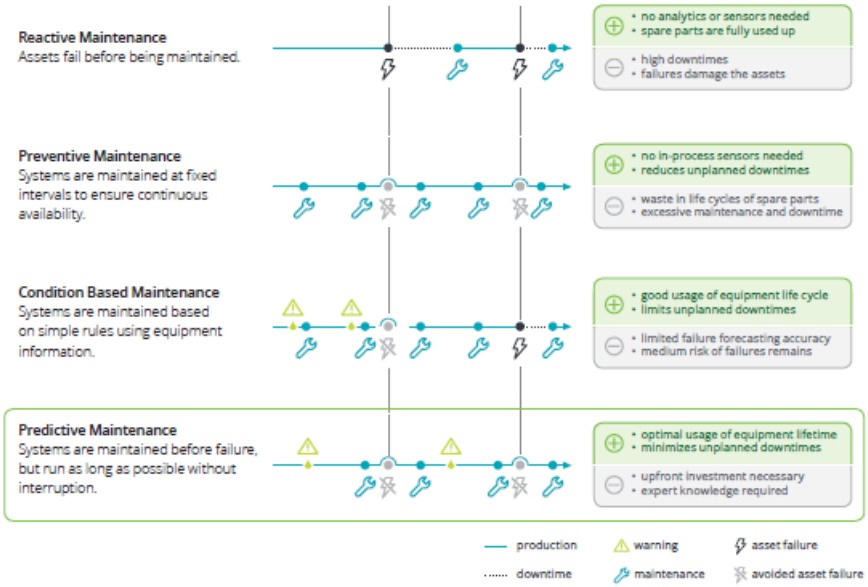


Figura 1.1: Strategie di Manutenzione in un contesto industriale

Le attività correlate con il *CBM*, ma più in generale con tutta la *PdM* sono:

- Acquisizione dei dati
- Preprocessing dei dati
- Definizione delle decisioni sulla manutenzione[8]

1.2.1 Acquisizione dei Dati

I dati vengono collezionati da differenti sorgenti e salvati su un supporto fisico. Generalmente sono i sensori che producono dati, di solito sotto forma di segnali inviati real-time. I vantaggi portati dall'utilizzo dei sensori sono in

contrapposizione con la loro complessità in termini di costo e sforzo dovuto all'installazione.

Ciò è in contrasto con quello che succede invece nella *Manutenzione Preventiva* dove le decisioni vengono prese non sulla base dei dati dei sensori generici, ma sulla base dell'esperienza del personale esperto o sistemi di allarme o logs di operatori.

I dati relativi ai singoli guasti[8] sono una parte fondamentale, cioè sono dati che indicano sotto quali circostanze si sia verificato un problema. Per estrarre questo tipo di informazione, vengono utilizzati meccanismi di rivelazione, identificazione e isolamento dei guasti.

1.2.2 Preprocessing dei Dati

Il passo successivo è elaborare i dati collezionati: i dati vengono estratti dal supporto in cui sono memorizzati e lavorati per eliminare quelli considerati superflui e per prepararli alla successiva elaborazione.

1.2.3 Definizione delle Decisioni sulla Manutenzione

Una volta ottenuti i dati puliti e organizzati, devono essere prese le decisioni riguardanti le policies di manutenzione delle risorse. Attraverso la diagnostica, ogni guasto viene misurato, memorizzato e identificato[8], ciò permette poi ai sistemi di *PdM* di sfruttare tutto per calcolare la *RUL* ovvero *Remaining Useful Life*, cioè una stima sul tempo che rimane prima che un guasto si verifichi[9].

Un indicatore essenziale e necessario per il calcolo della *RUL*[10] è l'introduzione della condizione attuale di una risorsa e della degradazione del suo segnale che poi porterà ad un guasto.

La sempre maggiore quantità di dati a disposizione, elemento fondante dell'IoT, ha per cui portato più attenzione a questa tipologia di argomento, ora più che mai di principale interesse per le aziende.

1.3 Benefici

Un importante indicatore quando si tratta di industria è l'*Overall Equipment Effectiveness* indicato con la sigla *OEE*[11]. L'*OEE* è un indice che riguarda il livello di accessibilità e le performance produttive di una data risorsa e la qualità dell'output prodotto[5]. I tempi di inattività non pianificati dovuti a guasti o a manutenzione hanno un impatto negativo sull'*OEE*; la *PdM*, nell'obiettivo di ridurre questi tempi, può avere un effetto positivo sull'*OEE*, inoltre anche la qualità dell'output ne può beneficiare.

La qualità di un prodotto non dipende solamente dallo stato di degradazione della macchine di produzione, ma anche, tra gli altri, dai componenti della macchina di produzione[12][9]. Un alto fattore di costo per le aziende è la necessità di pianificare la quantità di pezzi di ricambio necessari in un dato periodo di tempo, ciò comporta alti costi di inventario.

La *PdM*, pertanto, fornisce metodologie che permettono di mantenere un livello di pezzi di ricambio in inventario almeno pari a quanto viene predetto. In pratica, la *PdM* spesso manca di ottimizzazione congiunta con l'inventario, anche se sono stati creati modelli ottimizzati, ad esempio da Soltani[9].

La *PdM* offre anche benefici importanti nell'ambito del rilevamento da remoto per mantenere costanti le procedure di manutenzione anche in ambienti difficili e non sicuri per l'uomo.

1.4 Modellazione

Ottenuta la disponibilità dei dati lavorati è necessario occuparsi della creazione di un modello statistico utile per la *PdM*. I modelli comprendono tecniche che sfruttano il *Machine Learning* e il *Data Mining*. Il *Data Mining* si occupa di studiare grandi quantità di dati per cercare patterns e relazioni tra di essi.

Il *Machine Learning* invece, sfrutta questi dati e pattern per imparare come le variabili si influenzino a vicenda. La figura 1.2 identifica i vari livelli di complessità per quanto riguarda i modelli di *PdM*.

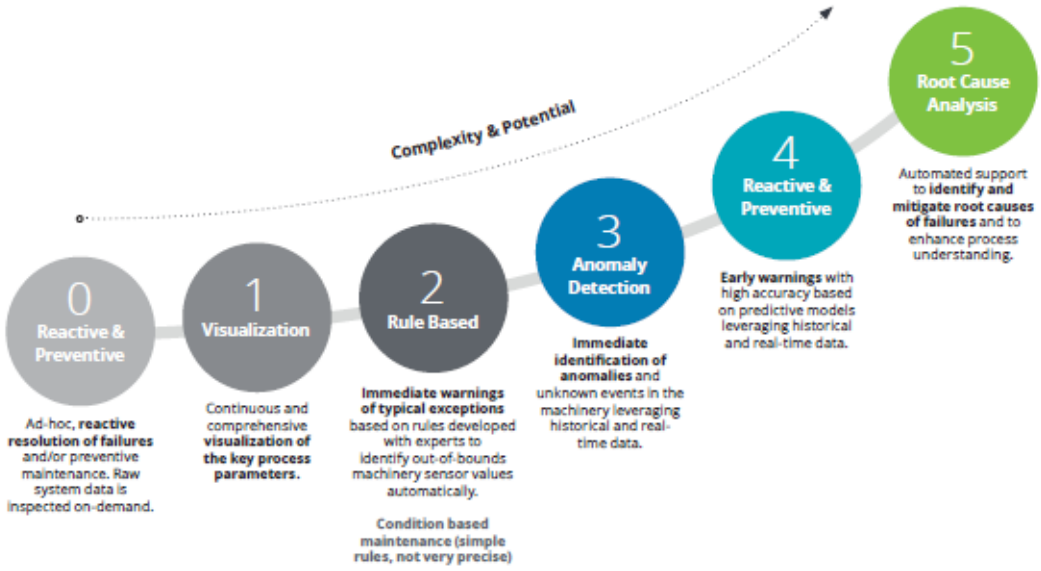


Figura 1.2: Livelli di complessità della Manutenzione Predittiva

1.4.1 Metriche Utilizzate

Tutte queste tecniche tipicamente lavorano e modellano la graduale degradazione delle risorse in un ambiente di *PdM*, dove la *RUL* è il principale indicatore. Una misura della degradazione può essere inclusa in un modello per la predizione, che poi sarà valutato sulla base del modello target.

Ogni modello di *PdM* deve essere valutato sulla base della sua abilità di predire il guasto di risorse, ma anche di non predire falsi allarmi. Le unità di misura più utilizzate per calcolare la qualità di un modello sono:

- **Precision**: $\frac{TP}{TP+FP}$
- **Recall**: $\frac{TP}{TP+FN}$
- **F1-Score**: $\frac{precision \cdot recall}{precision+recall}$

La *Precision* è calcolata dividendo le predizioni corrette (True Positive - TP) con la somma tra le corrette e i falsi positivi(False Positive - FP); in questo modo la *Precision* calcola quante predizioni proposte sono state effettuate correttamente.

La *Recall* è il risultato della divisione tra le predizioni corrette e tutte le predizioni fallite; così viene calcolato quanti fallimenti sono stati selezionati.

L'*F1-Score* calcola la media armonica tra Precision e Recall, così da capire il bilanciamento tra di essi.

I modelli di *PdM* sono strettamente dipendenti dalla strategia di manutenzione, la quale può essere influenzata da differenti fattori quali il costo di una riparazione o dalle conseguenze di un guasto. Perciò, tenendo conto queste influenze, *Precision* e *Recall* devono essere mantenute in equilibrio per ottenere una ottimizzazione.

1.4.2 Algoritmi di Modellazione

Il set di possibili algoritmi di *Data Mining* applicabili nella *PdM* è molto vario: la rilevazione di anomalie(punti di picco o cambiamenti di trend) è uno dei più ricorrenti in questo ambito in quanto si occupa di ricercare anomalie in serie temporali, che spesso possono coincidere con guasti.

Altro algoritmo che viene sfruttato è quello che si occupa di creare un modello di regressione. Infatti la regressione permette di trovare una relazione tra più variabili associandogli un valore numerico che in questo caso corrisponderebbe alla *RUL*.

Queste e altri algoritmi possono essere sfrutta nella *PdM*, ma sono stati citati questi due in quanto sono quelli utilizzati in questa tesi.

1.5 Sfide della Manutenzione Predittiva

In questa sezione verranno affrontati limiti e sfide della *PdM*, in questa lista sono presenti: ostacoli finanziari e organizzativi, limiti delle sorgenti dati, limiti delle attività di riparazione delle macchine e limitatezza dell'ottimizzazione.

1.5.1 Ostacoli Finanziari e Organizzativi

Da un punto di vista aziendale, la *PdM* è vista come investimento da cui poi trarre profitto. Per questa e altre ragioni devono essere valutati i costi e i benefici che la *PdM* può portare ad una azienda. Infatti gli sforzi che essa comporta, tra i quali installazione di sensori, estrazione di informazioni, preparazione di modelli e attività di manutenzione mirate, generano un costo per le compagnie. I costi ovviamente possono variare a seconda della complessità del dominio, del costo delle consulenze e dei costi di installazione ed estrazione. La figura 1.3 mostra come un investimento nella *PdM* possa portare benefici anche nell'immediato.

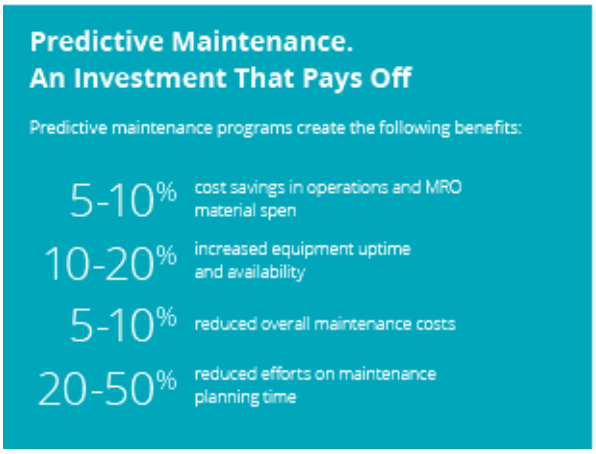


Figura 1.3: Investimenti nella Manutenzione Predittiva

Un metodo per valutare quando la *PdM* possa portare dei benefici è prevedere la creazione di un documento chiamato *ROI* ovvero *Return on Investiment*[1] che si occupa di valutare risultati della *PdM* in relazione ai suoi costi. Le logiche finanziarie sull'utilizzo e sull'applicabilità di *PdM* sono strettamente collegate alle dimensioni e al tipo di azienda, in cui è stata introdotta. Le medio-piccole aziende, di solito, sono più limitate dal punto di vista tecnologico, mentre le aziende più grandi possono essere meno influenzate dal rischio finanziario. I fornitori di tecnologia di solito si adattano alla scelta dei loro clienti e in base al mercato.

L'estrazione di insights dai dati relativi alla *PdM* è un ulteriore costo che deve essere valutato. Il monitoraggio continuo con informazioni correnti è disponibile, ma la cosa più complicata è capire in modo corretto l'output e farne una giusta visualizzazione[8]. Tutto ciò può portare a modelli non corretti, quindi ad allarmi non previsti e a manutenzioni programmate non necessarie.

1.5.2 Limiti delle Sorgenti Dati

Avere a disposizione sempre dati rilevanti nella creazione di modelli di *PdM* è essenziale. Tuttavia le aziende dichiarano raramente di essere in possesso di tutti i dati utili per una corretta introduzione alla *PdM*[1]. Dopo aver prelevato i dati disponibili, è quindi necessario ridurre il gap ricercando i dati rimanenti.

La qualità dei dati è un altro punto fondamentale che può portare a risultati non soddisfacenti nel caso in cui non sia sempre rispettata. E' necessario per cui capire se sono presenti sorgenti dati che forniscono informazioni non corrette o imprecise, per intervenire il più veloce possibile.

Per cui, la sfida tecnologica, consiste nel rapportarsi con i sensori, cercare di capire il caso in cui producano dati non esatti e sapere che anche essi sono soggetti a interruzioni e degradazioni[8].

1.5.3 Limiti delle Attività di Riparazione delle Macchine

Potendo prevedere il *RUL* di una risorsa, è possibile determinare i tempi di manutenzione, ma l'effettiva manutenzione di una risorsa deve ancora affrontare sfide legate alla sua dipendenza dalle interazioni umane e alla mancanza di auto-manutenzione.

Una risorsa dipende correntemente dall'operatore umano per procedure di controllo e manutenzione, per cui, dalla qualità e dalle skills dell'operatore, dipende l'efficacia di una procedura[13].

Le macchine di solito lavorano in reazione a comandi dati da uomini oltre che alla normale attività pianificata; l'attività degli operatori dipende dalla loro esperienza e dalle informazioni che hanno a disposizione. Avere a disposizione una risorsa intelligente, significherebbe prevedere azioni che le apportano benefici automaticamente. Un ulteriore passo verso l'autonomia delle risorse si basa sull'auto-consapevolezza e sull'auto-manutenzione delle stesse[13].

Una risorsa auto-consapevole può valutare le proprie condizioni, in relazione ai dati prodotti, e riconoscere condizioni critiche. In opposizione ad avere un unico sistema centrale che controlla tutti gli asset, c'è la necessità che tutte le informazioni per prendere decisioni, come ad esempio il modello di predizione, siano distribuite a tutti i livelli delle risorse. Le macchine dovrebbero poi pianificare le manutenzioni autonomamente a gruppi. Tuttavia il livello tecnologico ora presente nelle aziende non raggiunge questa auto-consapevolezza e auto-manutenzione.

1.5.4 Limitatezza dell'Ottimizzazione

Una decisione adeguata sul momento più appropriato per la manutenzione di una risorsa dipende dall'obiettivo e dagli indicatori che si sforzano di essere ottimizzati. Si parla di limitatezza della *PdM* in quanto ancora ci sono aspetti che i modelli non considerano, tra i quali:

- Attenzione all'ambiente
- Attenzione allo stoccaggio di materiali
- Inclusione dei costi delle strategie di manutenzione

Attenzione all'Ambiente

Al giorno d'oggi, l'impatto ambientale di una innovazione è un aspetto molto rilevante in ogni azienda[8]. Ciò non riguarda solamente l'energia

consumata, ma anche l'utilizzo di materiali grezzi. L'impatto ambientale è un criterio però che la *PdM* ancora non tiene in considerazione

Attenzione allo Stoccaggio di Materiali

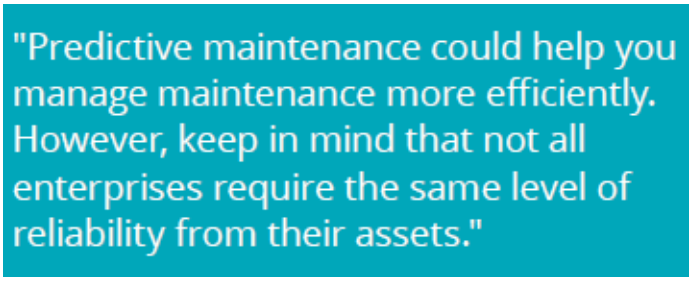
Un fattore di di successo tra le aziende è quanto bene una azienda è in grado di mantenere i suoi livelli di stoccaggio. Come già citato nella sezione 1.3, le metodologie di *PdM* spesso mancano di sviluppo simultaneo con il livello di inventario nel magazzino, che può portare a predizioni di manutenzione non perfette a livelli di stoccaggio non adeguati.

Inclusione dei Costi delle Strategie di Manutenzione

Una ulteriore mancanza nei modelli di *PdM* è l'inclusione del costo di definizione delle strategie di manutenzione all'interno dei modelli[2]. Oltre alle già citate *Precision*, *Recall* e *F1-Score* dovrebbero essere inclusi anche i costi delle strategie. Ad esempio: costo per una riparazione, tempo di inutilizzo di una risorsa durante una manutenzione etc... .

1.6 Conclusione

In questo capitolo è stato affrontato, in modo ampio, cosa sia la *Manutenzione Predittiva*, ma più in particolare quali siano i limiti e le sfide che essa deve affrontare per diventare una pratica sempre più affidabile e utilizzata dalle aziende. Sono stati elencati, nella sezione 1.5, le varie sfide che ancora limitano l'affidabilità e il successo della *PdM* tra le compagnie manifatturiere.



Non è possibile trarre conclusioni generali a riguardo dell'affidabilità della *PdM* in quanto ogni azienda si differenzia dalle altre. I fattori che potrebbero portare all'insuccesso sono vari: grandezza e organizzazione aziendale, prontezza tecnologica, disponibilità delle risorse e volontà di accettare i costi necessari. Si prevede che nel futuro prossimo nuove soluzioni si presenteranno e l'affidabilità sarà ulteriormente rafforzata. E' importante capire le influenze di questo nuovo campo dello sviluppo ovvero la *Manutenzione Predittiva* e di come supportarla per progredire ulteriormente.

Capitolo 2

Caso di Studio

2.1 Introduzione

In questo capitolo verrà preso in esame e approfondito il caso di studio oggetto di questa tesi progettuale. Come già citato nell'introduzione, le aziende partecipanti sono due: *SPOT Software* e *Cefla*. *SPOT* è stata l'azienda di riferimento che ha commissionato il progetto ed è stata intermediario con *Cefla*, in particolare con un suo reparto denominato *Finishing*, il quale è stato vero e proprio "cliente".

L'ambito di questa tesi riguarda l'applicazione di tecniche di *Manutenzione Predittiva* su dati provenienti da macchine che si occupano di eseguire stampe su legno. L'obiettivo, per cui, è quello di riuscire a fornire a *Cefla Finishing* uno strumento ulteriore, in aggiunta al classico applicativo già in uso, per riuscire a ridurre i costi derivanti dalle manutenzioni.

Quindi *SPOT*, che è la creatrice dell'applicativo che *Cefla Finishing* già utilizza, ha effettivamente commissionato il progetto. Qui di seguito verrà approfondito, chi sono le aziende partecipanti e, nel dettaglio, il caso di studio.

2.2 Aziende Partecipanti

Qui di seguito verranno presentate le due aziende partecipanti: chi sono e di che cosa si occupano.

2.2.1 SPOT Software

SPOT è una azienda di Cesena che si occupa di progettare e sviluppare App e Software innovativi, personalizzati e on-demand che integrano processi e dati, strumenti e tecnologie, prodotti e persone e li rendono sempre più intelligenti, interconnessi ed efficienti.

Sono una delle aziende più innovative e all'avanguardia per quanto riguarda l'utilizzo delle nuove tecnologie digitali e in particolare il servizio principale che offrono è quello di creare software ad-hoc per ogni cliente, a seconda delle sue esigenze.

Gli ambiti di cui si occupano svariano dall'IoT al Cloud, da App Mobile a System Integration e Industria 4.0.

Collaborano con molte delle principali grandi aziende del territorio tra cui, ovviamente *Cefla*.

Figura 2.1: Logo di Spot Software

2.2.2 Cefla Finishing

Cefla è una delle maggiori aziende dislocate nel territorio, operante anche a livello internazionale. Si compone di 5 diverse business units che mettono insieme competenze e capacità per il raggiungimento degli obiettivi nei rispettivi contesti, unite da un progetto comune in cui reti di relazioni e talenti si integrano e sostengono reciprocamente.

La business unit di riferimento di questa tesi è *Cefla Finishing*, la quale si occupa di creare impianti industriali per la verniciatura su differenti materiali.

Si occupano, per cui, di progettare e realizzare linee su misura per il mercato del legno, del vetro, della plastica, del fibrocemento, dei materiali compositi e del metallo, migliorando i processi industriali di molti settori. Sono all'avanguardia in questo settore anche perché dotati di un laboratorio di ricerca e sviluppo molto avanzato.

Figura 2.2: Logo di Cefla Finishing

2.3 Contesto

Prima di scendere nel dettaglio dei componenti di questo caso di studio, è necessario contestualizzare l'ambito in cui ci si trova. Il titolo della tesi rimanda già al settore manifatturiero, la sottosezione 2.2.2 evidenzia il sottosettore di riferimento ovvero gli impianti di verniciatura.

L'esigenza di un progetto di questo tipo nasce, non da una richiesta esplicita del cliente, ma da una volontà di *SPOT* di voler ulteriormente ampliare la propria offerta con una soluzione che sfrutti il *Machine Learning* e le moderne tecniche di previsione.

Infatti la soluzione proposta si posizionerebbe come ulteriore ausilio all'interno del gestionale che già il cliente possiede. Tutti le macchine per la verniciatura di materiali fanno parte di una cosiddetta "linea di produzione" ovvero il materiale grezzo entra nella prima macchina e di seguito viene passato alle altre, ognuna con un proprio compito, fino ad arrivare al prodotto finale.

Riuscire a ridurre gli sprechi e programmare più correttamente le manutenzioni, permetterebbe di ridurre i costi e ottimizzare la linea di produzione.

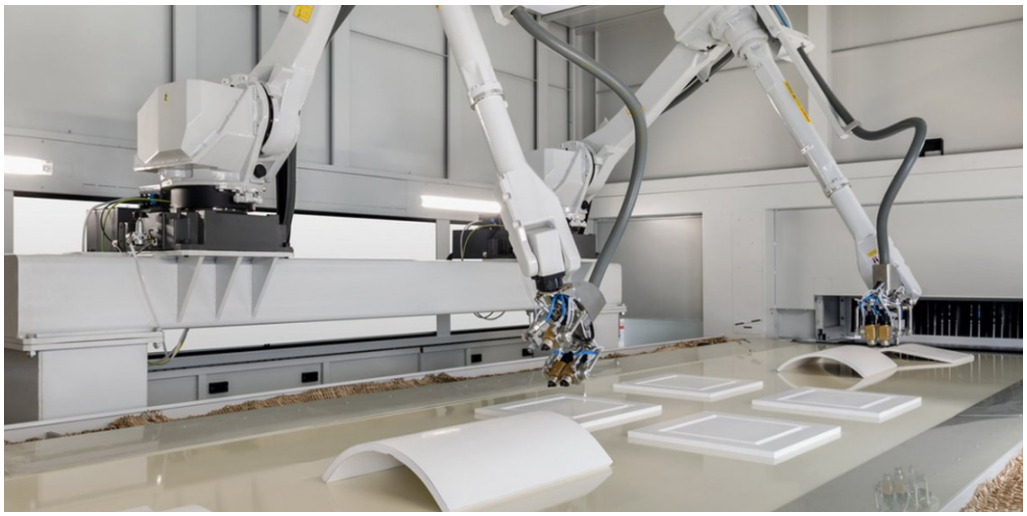


Figura 2.3: Esempio di macchina per la verniciatura

2.4 Panoramica

Il punto di partenza di questo caso di studio è stato descritto da *SPOT* nelle prime riunioni effettuate. Come già detto in precedenza, l'obiettivo è quello di analizzare i dati provenienti da macchine per la verniciatura e cercare di capirne il comportamento e sulla base di ciò, riuscire a riorganizzare gli interventi di manutenzione per ridurre al massimo gli sprechi.

Da qui, sono stati forniti i dati, contenuti all'interno di un classico database relazionale, su cui è stato effettuato tutto il lavoro di analisi e predizione.

Oltre alla tabella principale, è stata fornita una tabella delle manutenzioni, purtroppo non utile ai fini della predizione in quanto con pochi dati, ma utile per capire la classificazione delle manutenzioni e come sono strutturate, in modo tale da predisporre la soluzione quando saranno disponibili quei dati.

2.4.1 Database

I dati sono stati messi a disposizione su un unica tabella con 11 campi, essi si riferiscono ad un periodo di tempo di sei mesi che va, all'incirca, dal 15 Settembre 2018 al 15 Marzo 2019. I campi della tabella sono i seguenti:

- **Id**: identificatore univoco della tupla nella tabella, chiave primaria.
- **VariableId**: identificatore della singola variabile, variabili con lo stesso nome hanno lo stesso *VariableId*.
- **Timestamp**: istante di tempo in cui si è verificata la misurazione da inserire nel database.
- **TimestampEnd**: istante di tempo in cui si è concluso un allarme, per tutte le variabili che non sono allarmi è nullo.
- **Value**: valore della variabile in quell'istante di tempo.
- **StringValue**: campo non utilizzato perché presenta solo valori nulli.
- **Created**: istante di tempo in cui è stata inserita la tupla nel database.
- **CreatedByUserId**: campo non utilizzato perché presenta solo valori nulli.
- **VariableName**: nome della variabile.
- **VariableDeviceId**: macchina fisica che ha generato il valore della variabile.
- **VariableTypeId**: tipologia della variabile.

Da qui sono già evidenti i campi di principale interesse, che da qui in poi saranno i più utilizzati e citati, ovvero: *Timestamp*, *Value*, *VariableName*, *VariableDeviceId* e *VariableTypeId*.

L'oggetto del progetto vero e proprio sarà poi l'analisi delle singole serie temporali, per ogni variabile, per capirne il comportamento e l'andamento. Ciò verrà affrontato nei capitoli 4 e 5.

Num Record	21.778.818
Num VariableName Distinti	486
Num Allarmi Distinti	164
Num Segnali Distinti	322
Num Record Allarmi	12.254
Num Record Segnali	21.766.564

Tabella 2.1: Statistiche di base dei dati a disposizione

La tabella 2.1 mostra le statistiche di base riguardanti il numero totale di record nel database, il numero totale di allarmi e segnali e i loro record distinti,

2.4.2 Device e Types

Mentre *Timestamp*, *Value* e *VariableName* sono tre campi di cui è immediato il significato, è importante capire a fondo gli altri due campi, ovvero: *VariableDeviceId* e *VariableTypeId*.

VariableTypeId fa riferimento ad una classificazione delle variabili sulla base della loro tipologia, i tipi distinti di variabili sono quattro(numerati da 1 a 4) e si distinguono nel nome da differenti prefissi:

1. **Alarm**
2. **STA**
3. **RTM**
4. **PNL/CST**

La tipologia che più risalta è *Alarm* in quanto fa riferimento ad una condizione diversa rispetto alle altre tre tipologie di variabili che saranno identificate d'ora in poi come "segnali". Infatti gli allarmi possiedono un *Timestamp* di inizio e di fine e identificano una situazione critica.

Il campo *VariableDeviceId*, invece, identifica la macchina fisica vera e propria che ha generato la variabile: all'interno della tabella sono presenti 21 device distinti numerati da 1 a 31, con alcuni numeri mancanti. Le prime analisi effettuate sono state fatte sempre a partire da singoli device.

La tabella 2.2 mostra un esempio di tuple all'interno del database.

Timestamp	**Value**	**VariableName**	**VariableDeviceId**	**VariableTypeId**
2019-02-22 11:11:45	1.0	Alarm[1]	7	1
2018-10-17 08:03:36	2946.0	STA_PaintedPieces	16	2
2018-10-03 12:28:44	0.0	RTM_A301CurrentMean	2	3
2019-02-22 11:11:45	459.3	PNL_ExaustSmokeMean	1	4

Tabella 2.2: Esempio di tuple della tabella di riferimento

2.4.3 Correlazione Device-Alarm

Il device è un concetto fondamentale all'interno di questo caso di studio. Infatti una singola linea di produzione si compone di più macchine che tra loro collaborano e che tra loro si scambiano l'oggetto lavorato per arrivare al risultato finale. Un device corrisponde, quindi, ad una macchina all'interno di una determinata linea di produzione.

Quali allarmi un device possa generare è un aspetto di base che deve essere subito preso in considerazione all'interno di una panoramica generale. Queste informazioni sono state ricavate dalla tabella principale e poi confermate dal personale di *Cefla Finishing*.

La figura 2.4 mostra un grafo a dispersione che aiuta a capire la distribuzione degli allarmi su tutti i device: sull'asse x gli allarmi sono stati numerati da 1 a 164, mentre sull'asse y i device sono numerati con il loro identificato nella tabella. La colonna evidenziata mostra tutti i device che generano un allarme. Questo sarà poi il primo passo per fare una analisi di correlazione completa includendo anche le variabili e il loro andamento, oggetto dei capitoli 4 e 5.

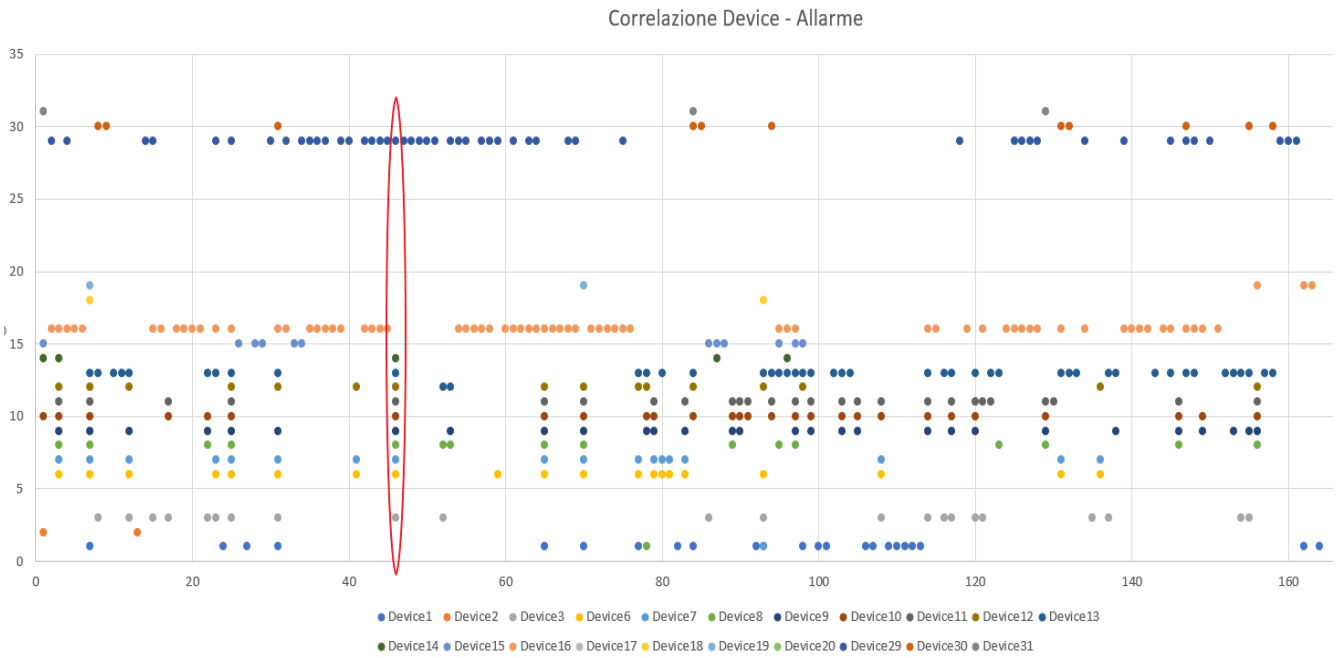


Figura 2.4: Grafo a Dispersione che mostra la correlazione Device-Alarm

D'altra parte anche il concetto di allarme è di primaria importanza in quanto, per definizione, identifica una situazione critica che dovrebbe essere il più possibile evitata. Capire la motivazione per cui un allarme si verifichi, analizzando le variabili ad esso correlate è un aspetto fondamentale per riuscire a prevenirlo.

2.5 Conclusione

In questo secondo capitolo, è stato presentato il caso di studio. In particolare, si è voluto mettere l'accento sul contesto e su una prima panoramica generale sulla situazione in cui ci si è trovati al momento di iniziare il progetto.

La naturale evoluzione di questo capitolo si trova in quelli successivi 4 e 5 in cui verrà presentato il progetto in termini di tecnologie utilizzate e di soluzione vera e propria.

Capitolo 3

Machine Learning e Previsione per Analitica Predittiva

3.1 Introduzione al Machine Learning

Per avere una giusta visione di cosa sia il *Machine Learning* è necessario definire nel modo corretto sia il termine stesso, sia cosa sia l'*Artificial Intelligence*, in quanto sono spesso associati e scambiati per la medesima cosa.

Come scritto da Bernard Marr in un articolo pubblicato su Forbes nel Dicembre 2016[14], l'*Artificial Intelligence - AI* può essere definita come un insieme di macchine che sono in grado di svolgere compiti(tasks) in un modo che il nostro mondo possa riconoscerle come "intelligenti"; il *Machine Learning* invece può definirsi come applicazione corrente dell'*AI* basata sull'idea che bisogna fornire alle macchine i dati e che da quelli poi vengano addestrate in modo tale che riconoscano determinati elementi senza che siano esplicitamente programmate.

3.1.1 Inizi

L'*Artificial Intelligence* come idea di base è presente da molto tempo, infatti i primi computer europei erano stati concepiti come "macchine logiche"

che riproducessero le basi aritmetiche e mnemoniche presenti nel cervello umano.

Col progredire della tecnologia e della conoscenza della nostra mente è cambiato il concetto di fondo dell'AI: piuttosto che calcoli sempre più complessi, il lavoro nel campo dell'AI si è concentrato sulla imitazione dei processi decisionali umani e sullo svolgimento di compiti in modi sempre più simili all'uomo.

I sistemi disegnati per l'AI sono spesso classificati in due gruppi:

- Applicati - Sistemi molto più comuni, come ad esempio sistemi di trading o di movimento autonomo dei veicoli.
- Generali - Sistemi che possono occuparsi di ogni tipo di task (compito), meno comuni, ma con molto più potenziale; è l'area in cui si è sviluppato il *Machine Learning*.

3.1.2 L'Ascesa del Machine Learning

Due importanti scoperte hanno portato alla nascita del *Machine Learning* come veicolo che sta portando avanti l'AI con la velocità che ha attualmente.

- La prima scoperta fu la realizzazione, accreditata da Arthur Samuel nel 1959, che, piuttosto che insegnare ai computer tutto ciò di cui hanno bisogno per conoscere il mondo e come svolgere i tasks, potrebbe essere possibile insegnarli a imparare da soli.
- La seconda, più recente, fu l'emergere di internet e l'enorme incremento di dati generati, immagazzinati e resi disponibili per l'analisi.

Una volta preso atto di queste innovazioni si pensò che sarebbe stato molto più efficiente codificarle per pensare come esseri umani per poi collegarle a internet.

3.1.3 Tipologie di Apprendimento

Si possono definire due grandi classi di metodi di apprendimento:

- **Apprendimento Supervisionato** - l'algoritmo di *Machine Learning* usa un dataset etichettato per dedurre l'esito desiderato. Questo però occupa notevole tempo e spazio, perché i dati necessitano di essere etichettati a mano.

- **Apprendimento Non Supervisionato** - non ci sono risposte predefinite o corrispondenti. L'obiettivo è ricercare i pattern nascosti tra i dati. Di solito viene utilizzato per clustering o task associativi, come raggruppare clienti per comportamento.

Mentre l'apprendimento supervisionato può essere utile, spesso, quando si lavora su ambiti meno conosciuti, si deve ricorrere a un apprendimento non supervisionato.

3.1.4 Reti Neurali

Le *Reti Neurali* sono state la chiave per permettere all'uomo di insegnare ai computer a pensare e a capire il mondo come noi lo intendiamo.

Una *Rete Neurale* è un sistema disegnato per classificare informazioni come il cervello umano. Come ad esempio riconoscere una immagine e classificare gli elementi che contiene.

Essenzialmente lavora su un sistema probabilistico in base ai dati forniti per addestrarla così da essere in grado di fare predizioni con un certo grado di accuratezza. Le applicazioni di *Machine Learning* possono variare dal riconoscere e valutare recensioni, a comprendere la melodia di un brano musicale, fino al riconoscimento di figure all'interno di immagini. L'idea di fondo di tutto ciò rimane il fatto di poter comunicare e interagire con le macchine come se interagissimo con altri esseri umani: il *Natural Language Processing - NLP* è una branca dell'*AI* in grande sviluppo che si occupa proprio di capire la comunicazione tra esseri umani sia scritta che parlata.

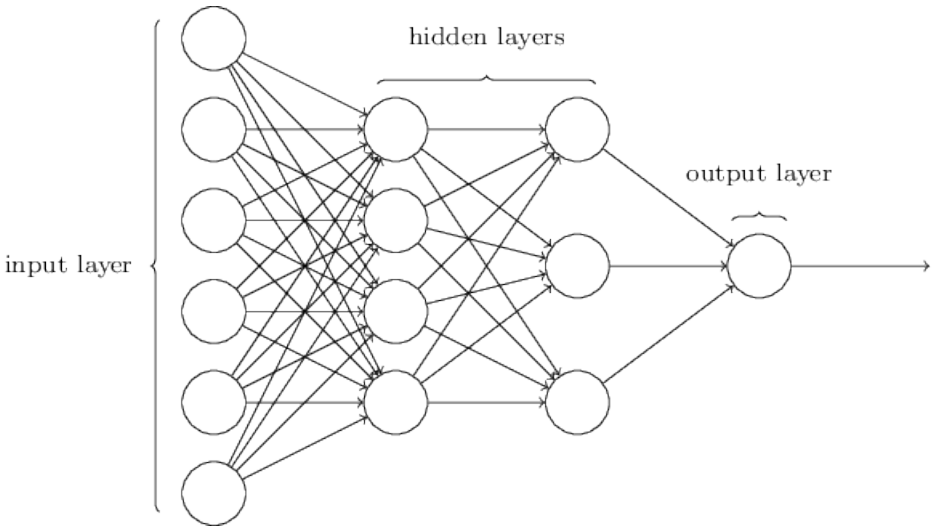


Figura 3.1: Rappresentazione di una Rete Neurale

Reti Ricorrenti

Le *Reti Neurali* possono essere rappresentate come grafi (con nodi e archi pesati), nei quali non ci sono cicli, come nel caso delle reti *Feed-Forward* oppure dove sono presenti cicli, che è il caso delle *Reti Ricorrenti*.

L'idea di fondo delle *RNN* è di utilizzare le informazioni sequenzialmente: in una *Rete Neurale* tradizionale si assume che tutti gli inputs e outputs siano indipendenti gli uni dagli altri, ma per alcuni tasks non è un approccio corretto. Infatti se ad esempio si desidera predire la parola successiva all'interno di una frase è importante sapere quale parola sia stata elaborata in precedenza. Le *RNN* sono chiamate "ricorrenti" perché eseguono lo stesso task per ogni elemento della sequenza in esame, con l'output che dipende dalla computazione precedente.

Si può pensare alle *RNN* come se avessero una "memoria" che catturi l'informazione su quello che è stato già calcolato. Teoricamente, le *RNN* possono utilizzare informazioni in sequenze arbitrariamente lunghe, ma in pratica sono limitate a guardare indietro solo di qualche step. Il grafo in figura 3.2 mostra una *RNN* che viene dispiegata in una rete completa. Dispiegarla significa semplicemente di riscrivere la rete nella sua sequenza completa.

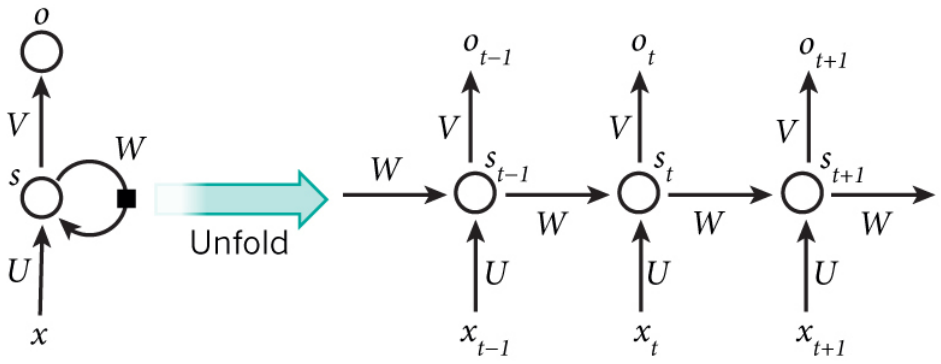


Figura 3.2: Rete RNN dispiegata, con parametri U,V e W

Le formule che regolano il calcolo in una *RNN* sono le seguenti:

- x_t è l'input allo step t.
- s_t è lo stato nascosto allo step t. è la "memoria" della rete; s_t è calcolato in base allo stato nascosto precedente e l'input allo step corrente corrisponde: $s_t = f(U_{x_t} + W_{s_{t-1}})$. La funzione f solitamente non è lineare. $s-1$, è tipicamente inizializzato a zero, perché è richiesto per calcolare il primo stato nascosto.
- o_t è lo step di output. Corrisponde a $o_t = softmax(V_{s_t})$.

Si può pensare allo stato nascosto s_t come la memoria della rete, il quale cattura l'informazione su cosa sia successo negli step precedenti. L'output o_t è calcolato solo sulla base della memoria allo step t.

A differenza di una *Rete Neurale* tradizionale, che utilizza diversi parametri ad ogni livello, una *RNN* condivide gli stessi parametri (U, V, W sopra) in tutti gli step. Questo riflette il fatto che si stia eseguendo lo stesso task ad ogni step, solo con diversi input. Ciò riduce notevolmente il numero totale di parametri che la rete deve imparare.

Il grafo in figura 3.2 produce output in ogni step, ma a seconda del task potrebbe non essere necessario. Allo stesso modo, potrebbe non avere bisogno di input a ogni step. La caratteristica principale di un *RNN* è il suo stato nascosto, che cattura alcune informazioni su una sequenza.

Le *RNN* hanno avuto notevole successo nella risoluzione dei tasks di *NLP*; la più utilizzata *RNN* è *LSTM - Long-Short Term Memory.* Alcuni esempi di utilizzo di *RNN* sono: modellazione del linguaggio e generazione automatica di testo, traduzione automatica di testo e riconoscimento vocale e di immagini.

La rete *LSTM* è quella che verrà presentata nella sezione 3.1.6 ed è la rete utilizzata per fare *Manutenzione Predittiva* all'interno del progetto di tesi.

3.1.5 Addestramento di una Rete Neurale

L'interesse principale delle *Reti Neurali* è per la possibilità di apprendimento e per l'addestramento. Dato uno specifico task da risolvere e una classe di funzioni $\boldsymbol{F}$, addestrare la rete significa usare un set di osservazioni per trovare $f^* \in \mathbf{F}$ che risolve il task in modo ottimale.

Ciò comporta la definizione di una funzione di costo $C : F \to \mathbb{R}$, cosicché la soluzione ottima f^* è $C(f^*) \leq C(f) \forall f \in F$, quindi nessuna soluzione costa meno di quella ottima.

La funzione di costo è un concetto importante nell'apprendimento, in quanto misura quanto lontano sia una particolare soluzione da una ottimale per il task da risolvere. L'algoritmo di apprendimento cerca nello spazio delle soluzioni per trovare una funzione che ha il minor costo possibile.

Per applicazioni dove la soluzione dipende dai dati, il costo deve necessariamente essere una funzione delle osservazioni, altrimenti il modello non è in relazione con i dati; ciò è spesso definito come una statistica a cui possono essere apportate solo approssimazioni. Ad esempio, considerando il problema di trovare il modello *f*, che minimizzi $C = E[(f(x) - y)^2]$, per una coppia (x,y) pescata dall'insieme $\mathcal{D}$. In una situazione pratica avremmo solamente N campioni da $\mathcal{D}$ e quindi si dovrà minimizzare

$$\widehat{C} = \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2 \tag{3.1}$$

Per cui il costo è minimizzato su un subset dei dati invece che su tutta la distribuzione; è necessario avere un modello che definisca in quale ambiente opera una *Rete Neurale*, questi modelli si possono definire come Paradigmi di apprendimento che distingue le regole e gli algoritmi per il corretto apprendimento.

3.1.6 Reti Neurali Long-Short Term Memory

L'apprendimento per memorizzare informazioni su intervalli di tempo prolungati attraverso backpropagation ricorrente richiede molto tempo, soprattutto a causa di un Backflow Error insufficiente. Hochreiter e Schmidhuber analizzano il problema e offrono una soluzione, efficiente e basata sul gradiente, quale la *Rete Neurale* di tipo *Long-Short Term Memory*[15].

Troncando il gradiente laddove è possibile farlo, *LSTM* può imparare a colmare il minimo ritardo di tempo, in eccesso di 1000 time step, applicando un flusso costante di errore all'interno di unità speciali (Gate Units). Le Gate Units imparano ad aprire e chiudere l'accesso al flusso di errore costante.

LSTM è una *Rete Neurale* ricorrente, locale nello spazio e nel tempo, ed ha una complessità computazionale per step e weight pari a $O(1)$; è inoltre in grado di risolvere tasks complessi, artificiale e con lunghi ritardi, che non erano mai stati risolti prima da una rete ricorrente.

Inizialmente, le reti ricorrenti, ad esempio le *Short-Term Memory*, utilizzavano le connessioni di feedback per immagazzinare informazioni degli input più recenti; ciò è potenzialmente significativo per molte applicazioni come il riconoscimento vocale, ma se lo scarto di tempo tra l'input e il segnale successivo non è più minimo, allora questo tipo di rete non funziona correttamente e in tempi brevi; in alternativa a ciò, le *LSTM*, cambiando i valori dei pesi lentamente, riescono a ovviare a queste problematiche. L'espressione *Long-Short Term* si riferisce al fatto che *LSTM* è un modello di *Short-Term* applicato a un lungo periodo di tempo.

Un blocco *LSTM* è composto da quattro componenti principali: una cella, un input gate, un output gate e una forget gate.

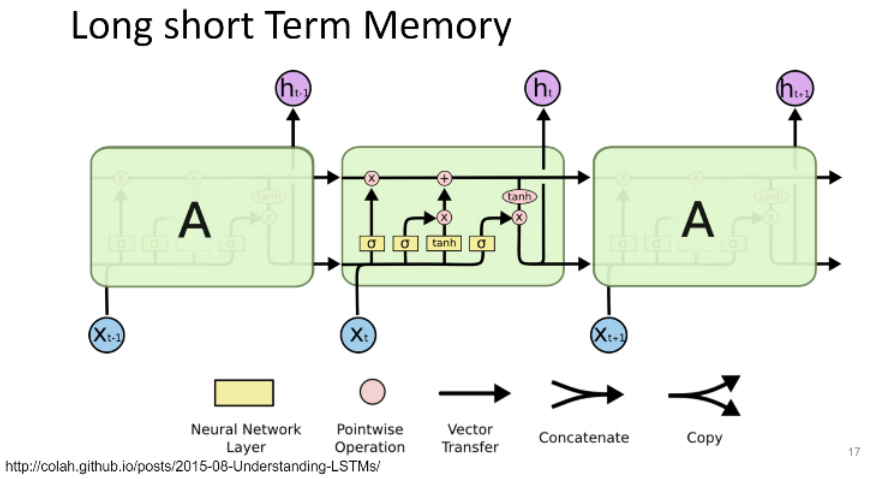


Figura 3.3: Rappresentazione di una LSTM

La cella è la responsabile per "ricordare"informazioni ogni intervallo di tempo arbitrario.

Ognuno dei tre gate può essere considerato come un neurone artificiale "convenzionali", come in una *Rete Neurale* multi-layer (o feedforward): cioè calcolano un'attivazione (utilizzando una funzione di attivazione) di una somma ponderata. Intuitivamente, possono essere considerati come regolatori del flusso di valori che attraversa le connessioni del *LSTM*; da qui la denotazione "gate". Ci sono connessioni tra questi gate e la cella. Alcune delle connessioni sono ricorrenti, alcune non lo sono. *LSTM* è adatto per classificare, processare o predire una serie di dati temporali di dimensione sconosciuta.

3.2 Introduzione all'Analitica Predittiva

L'*Analitica Predittiva*(o *Previsionale*) è una materia che comprende una varietà di tecniche statistiche riguardanti la modellazione predittiva, l'apprendimento automatico e il *Data Mining* per analizzare fatti storici e attuali e fornire predizioni sul futuro o eventi sconosciuti[16][17][18].

L'obiettivo primario è predire il futuro utilizzando i dati (serie storiche o altro) disponibili in questo momento: le previsioni sono alla base di decisioni che si prendono immediatamente.

La definizione di previsione ne comprende tre fondamentali:

- **Orizzonte Previsionale**: numero di periodi futuri coperti dalla previsione.
- **Periodo Previsionale**: di tempo per cui si fanno le previsioni.
- **Intervallo Previsionale**: frequenza con cui si effettuano nuove previsioni.

3.2.1 Modelli Previsionali

Quando si affronta un argomento come le previsioni è importante capire che nella maggior parte dei casi esse sono errate: una buona previsione, per cui, non fornisce solamente un numero esatto, ma anche valori medi, dispersione e un intervallo di affidabilità. Ci sono molti modelli che permettono di fare previsioni, che si dividono su due categorie:

- **Metodi Soggettivi**: sono metodologie basate su stime, sondaggi tra i clienti, opinioni dei dirigenti e metodo Delfi.
- **Metodi Oggettivi**: sono metodologie basate su modelli matematici, i più rinomati sono i modelli econometrici e le serie storiche, che richiedono varie attività, tra cui:
  - *Identificazione del modello (model specification)*: processo di selezione della tecnica previsionale da utilizzare.
  - *Identificazione parametrica (model fitting)*: dato il modello, stima i parametri in modo da rendere i risultati compatibili con i dati sperimentali.

– *Validazione del modello (model diagnosis)*: determina quanto il modello sia coerente con i dati sperimentali e quindi quanto le assunzioni fatte siano soddisfatte.

3.2.2 Serie Storiche

Definizione

Una serie storica (o serie temporale) è semplicemente un insieme di valori ordinati rispetto al tempo, esprime la dinamica di un certo fenomeno nel tempo. Modellisticamente, si suppone che esistano n osservazioni provenienti da altrettante variabili casuali dipendenti. Obiettivo dello studio delle serie storiche è l'individuazione di schemi nei dati passati, che si ipotizzano ripetersi anche nel futuro.

Un esempio di serie storica, preso direttamente da una delle variabili coinvolte nel progetto, è mostrato nella figura 3.4.

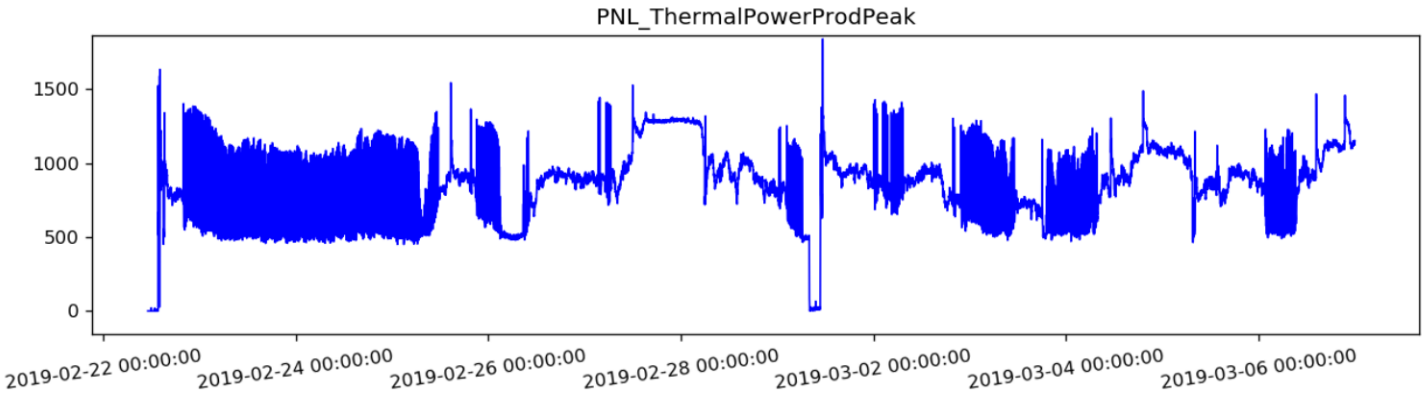


Figura 3.4: Serie Storica della variabile *PNL_ThermalPowerProdPeak*

Componenti

Una serie storica può comporsi di cinque componenti differenti:

- **Componente Tendenziale**(Trend) μ_t: crescita o diminuzione a lungo termine della serie, in cui il tasso di variazione è relativamente costan-

te. Tipicamente rappresentata con un modello lineare, quadratico o esponenziale.

- **Componente Stagionale** s_t: contiene pattern ripetitivi , ad esempio associati a variazioni climatiche o calendariali, individuabili su periodi annuali, mensili e settimanali.

- **Componente Ciclica** c_t: fluttuazioni congiunturali non spiegate da effetti di stagionalità, tipicamente dovute a variazioni delle condizioni economiche (recessioni, espansioni).

- **Componente Casuale** γ_t: piccole oscillazioni dovute ad eventi casuali.

- **Componente Occasionale**: componente rara, dovuta a eventi bellici, importanti innovazioni tecnologiche, crisi politiche etc... .

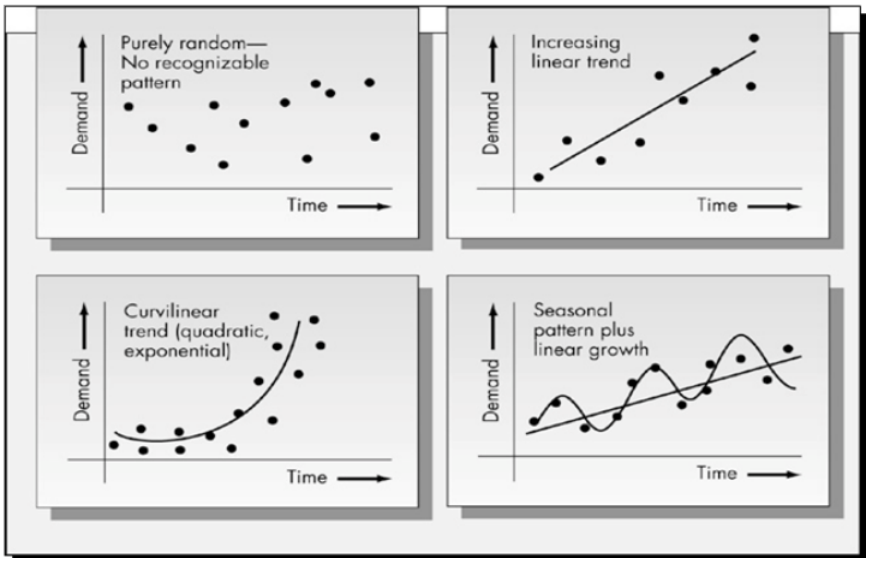


Figura 3.5: Componenti principali Serie Storiche

Modelli

Una serie storica si sviluppa lungo un intervallo di valori nel tempo, indicati sempre con il simbolo y_t. Si fa l'ipotesi che i valori osservati y_t nella serie storica siano il risultato della combinazione di componenti indicati in precedenza. Relativamente al modo in cui le componenti si combinano si distinguono modelli additivi, moltiplicativi e modelli misti.

- **Modelli Additivi**: $y_t = \mu_t + s_t + c_t + \gamma_t$
- **Modelli Moltiplicativi**: $y_t = \mu_t \times s_t \times c_t \times \gamma_t$
- **Modelli Misti**: $y_t = \mu_t \times s_t \times c_t + \gamma_t$

3.2.3 Metriche per il Calcolo degli Errori

Come già citato in precedenza, spesso le previsioni sono errate, quindi è importante associargli un errore che identifichi i possibili intervalli di previsione. Ovviamente, più si riduce l'errore calcolato, più la bontà della previsione sarà garantita. Ci sono diverse possibili metriche che possono essere utilizzate in questi casi, sia che si utilizzino modelli che sfruttano *Reti Neurali* o meno.
Sia $X = (x_1, \ldots, x_n)$ i dati osservati e $F = (f_1, \ldots, f_n)$ i dati previsti:

- **BIAS**: media aritmetica degli errori $\rightarrow BIAS = \frac{\sum_{i=1}^{n}(x_i - f_i)}{n}$
- **MAD**(Mean Absolute Deviation): media degli errori assoluti $\rightarrow$ $MAD = \frac{\sum_{i=1}^{n}|x_i - f_i|}{n}$
- **MSE**(Mean Squared Error): media degli errori quadratici $\rightarrow$ $MSE = \frac{\sum_{i=1}^{n}(x_i - f_i)^2}{n}$
- **Standard Error**: $\sqrt{MSE}$
- **MAPE**(Mean Absolute Percent Error): media degli errori assoluti in percentuale $\rightarrow MAPE = \frac{\sum_{i=1}^{n}\frac{|x_i - f_i|}{f_i} \cdot 100}{n}$

3.3 Machine Learning per Previsione di Serie Storiche

L'*Analitica Predittiva* riguarda fondamentalmente la previsione di serie storiche in tutte le sue possibili declinazioni. Le previsioni possono essere fatte sfruttando un approccio stocastico, molto sviluppato in letteratura (modelli ARIMA, SARIMA etc...) che permette di avere risultati ben motivati oppure un approccio neurale ovvero utilizzando il *Machine Learning* e le *Reti Neurali*, approccio molto più recente che fornisce mediamente risultati migliori, ma senza giustificazioni ai modelli sviluppati.

L'approccio che verrà descritto sarà quello neurale in quanto sfruttato nel progetto di questa tesi. Le *Reti Neurali - NN* sono uno strumento molto efficace. Infatti riescono a gestire naturalmente modelli non lineari, di solito i dati in ingresso sfruttati da *NN* sono generati da processi complessi e non lineari. La forza delle *NN* sta nel generare e combinare modelli lineari e non con qualunque livello di accuratezza richiesto:

- Possono apprendere pattern in serie temporali lineari
- Possono apprendere pattern in serie temporali non lineari
- Possono generalizzare pattern lineari e non lineari

La sottosezione 3.1.4 mostra nel dettaglio la struttura di una *NN* ed il suo funzionamento, l'addestramento viene invece affrontato nella sottosezione 3.1.5.

Le *NN* sono in grado di approssimare una qualunque funzione; la previsione può essere vista come l'identificazione di una particolare funzione (non lineare) e sono in grado di supportare vari tipi di previsione: serie storiche, causali, combinate.

3.3.1 Modello per Reti Neurali

Per definire un modello di previsione neurale generalmente sono 5 le fasi che devono essere seguite indipendentemente dalla tipologia di rete che viene utilizzata. Le fasi sono le seguenti:

1. Preprocessing (uguale ai modelli statistici)
    - Trasformazioni(diff, log,...)
    - Scaling, se sono coinvolte variabili diverse le si porta a valori comparabili
    - Normalizzazione
2. Definizione dell'architettura della *NN*
    - Numero di neuroni in ingresso, nascosti, di uscita
    - Numero di strati nascosti
    - Elaborazione dei nodi (funzione di attivazione)
    - Interconnessione fra livelli
3. Training
    - Inizializzazione dei pesi
    - Algoritmo di training
    - Parametri di training
4. Utilizzo della *NN*
5. Validazione
    - Scelta del dataset
    - Criteri di validazione

Fondamentale è la scelta del dataset e soprattutto la qualità dei dati proposti. E' importante una giusta scelta delle dimensioni di Train e Test Set (di solito 80% Train Set e 20% Test Set). Tutte queste fasi sono state utilizzate nel progetto e affrontate nei successivi capitoli.

3.3.2 Sliding Window

Le *NN*, per riuscire a calcolare l'output il più corretto possibile, sfruttano una tecnica chiamata *Sliding Window*, la quale non da alla *NN* tutta la serie storica nello stesso momento, ma suddivide la serie storica in finestre successive e una alla volta viene passata alla *NN*. L'apprendimento si sviluppa per cui in 5 passi:

1. In ingresso alla *NN* viene data una finestra prefissata di dati storici.
2. La *NN* calcola l'output relativo a quella finestra.
3. L'output viene confrontato con il dato reale corrispondente.
4. Si attiva la funzione di *Backpropagation* ovvero si cambiano i pesi in modo da ridurre l'errore nella previsione.
5. Si sposta la finestra a quella successiva e i passi vengono ri-eseguiti fino alla consumazione totale della serie storica.

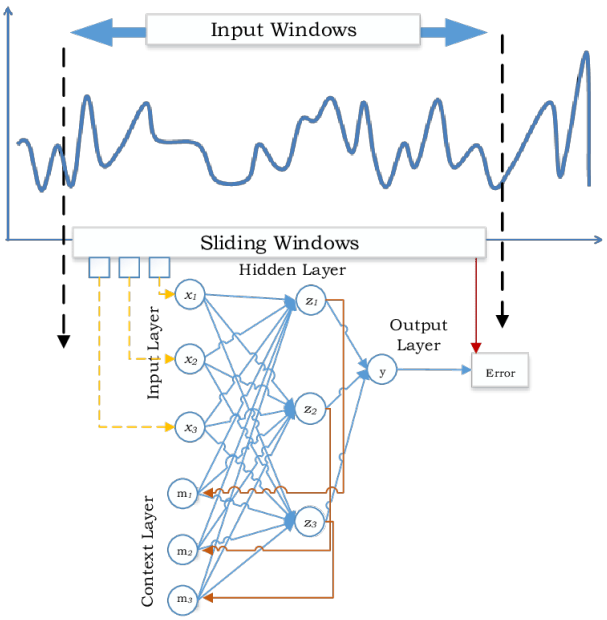


Figura 3.6: Esempio di utilizzo della Sliding Window

3.3.3 Tecnologie per Previsione

Le tecnologie che permettono di utilizzare le *NN* per fare previsioni di serie storiche, in tutte le sue possibili declinazioni, sono differenti e dipendenti da differenti linguaggi. In particolare, in questo progetto di tesi, i linguaggi utilizzati sono stati due: *C#* e *Python*.

Per quanto riguarda invece le tecnologie, approfondite poi nel capitolo 5, sono state utilizzate le seguenti:

- **Tensorflow**: libreria scritta in *Python* che permette di creare la propria *NN*, scegliere le metriche, fare validazione e testing.
- **Microsoft ML**: strumento di *Microsoft* che permette di eseguire tutte le procedure classiche(sentiment classification, binary classification, regression ...). Questo modulo permette di specificare input e autonomamente calcola l'algoritmo più efficace e restituisce l'output con tutte le metriche del caso. *Microsoft ML* è un modulo ampio che comprende altre diverse procedure come il calcolo di punti di *Spike Points* e *Change Points*, utilizzato in questa tesi.

3.4 Conclusioni

In questo capitolo sono stati presentati il *Machine Learning* e l'*Analitica Predittiva* e come le due materie si uniscano per riuscire a fare previsione in relazione a serie storiche. Le serie storiche sono un argomento centrale di questa tesi in quanto le variabili, che sono l'elemento preponderante dello studio e della sperimentazione, sono tutte associate a serie storiche. Nei successivi capitolo si entrerà più nel merito delle tecnologie e delle tecniche utilizzate.

Capitolo 4

Schema e Moduli del Progetto

4.1 Introduzione

I precedenti capitoli hanno mostrato una panoramica generale su tutto il mondo della *Manutenzione Predittiva - PdM* e hanno introdotto a quello che è il caso di studio preso in esame in questo progetto e le tecnologie impiegate.

Ora è il momento di scendere nel dettaglio del progetto, prima dal punto di vista più teorico(questo capitolo) e poi dal punto di vista più pratico e tecnico(capitolo 5).

Per cui, nelle successive sezioni, verranno analizzati i singoli moduli che compongono il progetto che hanno portato al raggiungimento dell'obiettivo che è l'applicazione di *PdM* su un caso reale.

I moduli sono cinque, alcuni utili direttamente per la *PdM*(sezioni 4.2, 4.4 e 4.6), mentre altri più collaterali, ma che ampliano e completano il progetto(sezioni 4.3 e 4.5).
I moduli sono i seguenti:

- Feature Selection e Analisi di Correlazione
- Qualità dei Segnali
- Individuazione Punti Significativi
- Predizione Trend

- Predizione Remaining Useful Life

La situazione di partenza è quella presentata nel capitolo 2, in cui l'intero dataset è stato messo a disposizione e analizzato. Da qui, sono stati realizzati tutti i moduli che verranno presentati.

4.2 Feature Selection e Analisi di Correlazione

Date le grandi dimensioni del dataset a disposizione, la prima cosa che è stato necessario eseguire è una selezione delle variabili più significative, detta anche *Feature Selection*. Dato che l'azienda cliente non è stata in grado di fornire le variabili più rilevanti, si è reso necessario anche una analisi di correlazione tra device-allarme-segnale.

Il punto di partenza di questo di analisi è stato concentrarsi sui singoli device: come già detto nel capitolo 2, i device sono le macchine vere e proprie che fisicamente svolgono le loro attività. I passi necessari alla *Feature Selection* sono stati:

1. Distinguere i segnali e gli allarmi.
2. Capire quali variabili ogni device sia in grado di generare.
3. Cercare di visualizzare le variabili che vengono generate entro 30 secondi dall'attivazione di un allarme.
4. Individuare le sequenze di variabili generate da ogni singolo device.
5. Sovrapporre la serie temporale di un segnale attivato da un device ad un allarme sempre da lui stesso attivato.

4.2.1 Generazione Variabili da Device

Già nel capitolo 2 i segnali e gli allarmi sono stati considerati separatamente, questa distinzione è individuabile grazie ad una semplice query. Per

quanto riguarda invece l'individuazione di quali variabili ogni device sia in grado di generare, è stata eseguita una query più approfondita che ha prodotto come risultato la tabella 4.1 che sintetizza a seconda della tipologia di variabile.

VariableTypeId	**VariableNamePrefix**	**GeneratedByDevice**
1	Alarm	All Except 17/20
2	STA	All
3	RTM	All Except 20
4	PNL/CST	1

Tabella 4.1: Tabella di sintesi per capire quali tipologie di variabili i device generino

Per leggere al meglio la tabella 4.1 è bene ricordare che i device sono 21 numerati da 1 a 31 con alcuni numeri mancanti. Subito risalta il fatto che i device 17 e 20 non producano allarmi, che ci permette già quasi di escluderli dalle successive computazioni, e che la tipologia 4 sia generata solo dal device 1, elemento che invece è più da studiare e da verificare con il cliente.

La tabella estesa, non presente graficamente qui, ha permesso di costruire il grafo a dispersione indicato nella figura 2.4.

4.2.2 Analisi delle Sequenze di Variabili

I punti 3 e 4 dell'elenco precedente sono stati sviluppati grazie ad alcuni script in *Python* che hanno permesso di analizzare quali variabili fossero generate nei 30 secondi successivi all'inserimento di un allarme nel dataset. Il tempo è indicativo in quanto è solamente un parametro ovviamente modificabile.

In questo modo si è stati in grado di capire e di avere una di visione di quali variabili potessero essere collegate tra di loro.

4.2.3 Correlazione Device-Segnale-Allarme

L'ultima parte comprende un lavoro che poi risulterà come l'input dell'individuazione dei punti significativi nella sezione 4.4.

Infatti, al sistema, è stato aggiunta la possibilità di selezionare un device, un segnale e un allarme e, da qui, mostrare un grafico di come l'allarme si sovrapponga alla serie temporale di un segnale. Questa visualizzazione permette di avere una prova immediata della sovrapposizione di un allarme nel momento in cui scaturisce un segnale.

La figura 4.1 mostra un esempio sfruttando le prime trenta occorrenze della correlazione tra il device 16, l'allarme 1 e il segnale *STA_DistTravelZ2*.

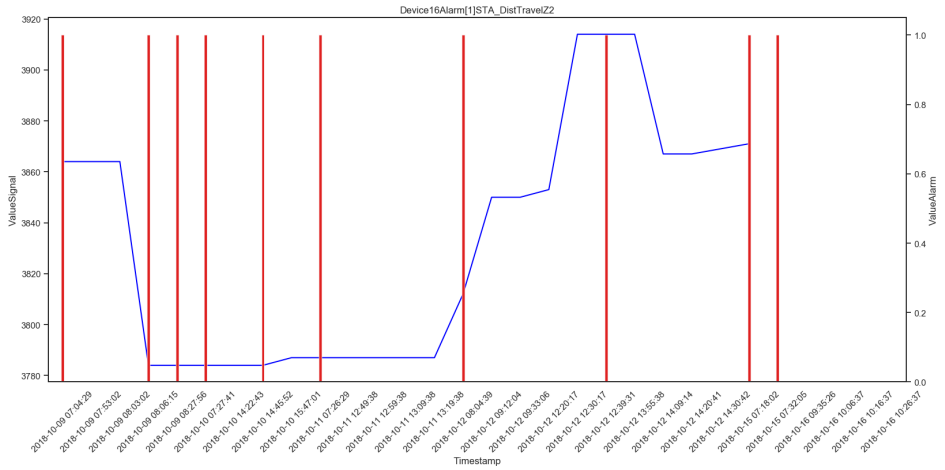


Figura 4.1: Esempio di correlazione tra device-allarme-segnale

4.3 Qualità dei Segnali

Questo modulo nasce con l'obiettivo di classificare l'ultimo record di un segnale, relativo ad un device, dandogli un ranking(da 1 a 5) che certifichi la "qualità" del segnale stesso sulla base dei dati precedenti.

Il ranking va da 1 che identifica il risultato migliore a 5 che è quello peggiore. Per "qualità", si intende quanto il valore corrente si discosti da quelli precedentemente presenti nel dataset.

Questa parte non è direttamente collegata al lavoro di *PdM*, ma è stata fatta sotto esplicita richiesta da parte del cliente.

Per svolgere questo compito sono stata utilizzata una statistica molto conosciuta ovvero la *Media Mobile* nelle sue due applicazioni: *Simple Moving Average - SMA* e *Exponential Moving Average - EMA*.

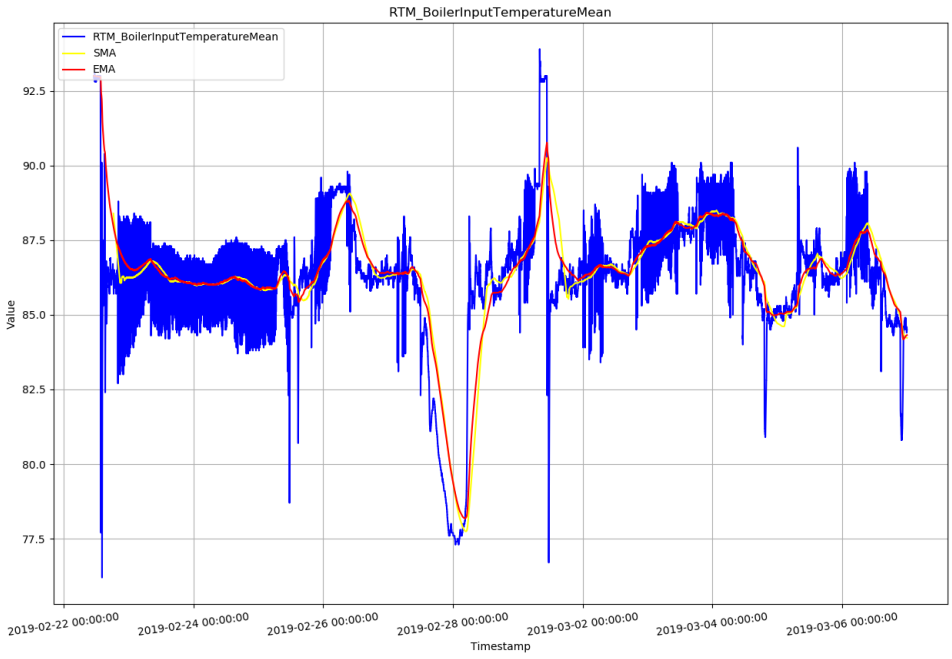


Figura 4.2: Esempio di SMA e EMA dato un segnale

4.3.1 Media Mobile - SMA e EMA

Definizione

Data una serie storica y_t, $t = 1, 2, \ldots, T$, contenente i valori osservati di una variabile Y dal tempo 1 al tempo T, siano:

- m_1 il numero dei periodi precedenti a t;
- m_2 il numero dei periodi successivi a t;
- θ_i il peso da attribuire all'i-esimo valore osservato;

Si definisce media mobile al tempo t il valore:

$$mm_t = \frac{1}{k} \sum_{i=-m_1}^{m_2} \theta_i y_t + 1 \tag{4.1}$$

dove $k = m_1 + m_2 + 1$ è il periodo o l'ordine della media mobile, ed equivale al numero degli addendi. In statistica, la media mobile è uno strumento utilizzato per l'analisi di serie storiche, in questo progetto permette anche di capire quale sia il trend di una data serie temporale.

SMA

Media mobile semplice, detta anche aritmetica, rimane quella più usata e di più facile calcolo. Vengono presi i dati di un determinato periodo e ne viene calcolata la media sommandoli fra loro e dividendo per il numero totale di valori.

E' importante sapere però che questo tipo di media però assegna la stessa importanza ad ogni singolo dato.

Nella figura 4.2 è colorata di giallo, il periodo è di tre giorni e la serie temporale originale è blu.

EMA

Questa media mobile esponenziale cerca di eliminare le carenze della SMA. Viene quindi dato un peso differente ai vari valori, maggiore ai più recenti e minore a quelli più vecchi, fatto che porta molti a definirla media mobile ponderata esponenziale.

Nonostante dia un'importanza minore ai valori passati li include ugualmente nel suo calcolo prendendo in esame quindi molti più valori di quelli definiti dal periodo della media mobile.

Nella figura 4.2 è colorata di rosso, il periodo è sempre di tre giorni.

4.4 Individuazione Punti Significativi

Modulo di centrale importanza all'interno del progetto di tesi alla pari con la predizione del *RUL*. Infatti questa parte ha permesso di individuare quelli che sono i punti di picco e i punti di cambio di tendenza e, grazie ad essi, chiudere il cerchio per quanto riguarda la correlazione device-allarme-segnale.

Infatti i punti significativi, detti anche anomalie, sono punti che, all'interno di una serie temporale, si differenziano dagli altri. Queste anomalie è importante riconoscerle attraverso una soluzione software perché spesso non basta la visione del grafico della serie temporale.

Ci sono diverse soluzioni che permettono di evidenziare queste anomalie: quella scelta è stata la proposta di *Microsoft* con il suo motore di *Machine Learning* detto *ML.NET*.

4.4.1 ML.NET

ML.NET offre la possibilità di aggiungere funzionalità di *Machine Learning* alle applicazioni .NET. Con questa funzionalità è possibile eseguire stime automatiche e altre operazioni usando i dati disponibili per l'applicazione.

Le applicazioni di *Machine Learning* usano i modelli nei dati per eseguire stime anziché dover essere programmate in modo esplicito. Centrale per *ML.NET* è un modello di *Machine Learning*. Il modello specifica i passaggi necessari per trasformare i dati di input in una stima. Con *ML.NET* è possibile eseguire il training di un modello personalizzato specificando un algoritmo oppure è possibile importare modelli *TensorFlow*.

Le operazioni effettuabili con *ML.NET* sono:

- Classificazione
- Regressione
- Rilevamento anomalie
- Suggerimenti di prodotti interessanti

- Predizione serie temporali
- Classificazione immagini

Ogni operazione possiede un proprio modello a cui i dati in input devono adattarsi per permettere l'esecuzione della stessa e il calcolo dell'output.

Nel nostro caso di studio, l'operazione che si è scelto di effettuare con *ML.NET* è il rilevamento della anomalie che ci è utile per trovare quali variabili generino determinati allarmi.

4.4.2 Rilevamento Anomalie

Il rilevamento delle anomalie segnala eventi o comportamenti inattesi o insoliti all'interno di serie temporali. Il rilevamento delle anomalie è il processo di definizione degli outlier; indica i punti in una determinata serie temporale di input il cui comportamento non è quello previsto o è "strano". Esistono due tipologie di anomalie: punti di picco(spike points) e punti di modifica(change points).

Spike Points

I picchi indicano comportamenti anomali nel sistema che risultano essere temporanei ovvero outlier notevoli durante un normale andamento, spesso sono punti visibili ad occhio nudo.

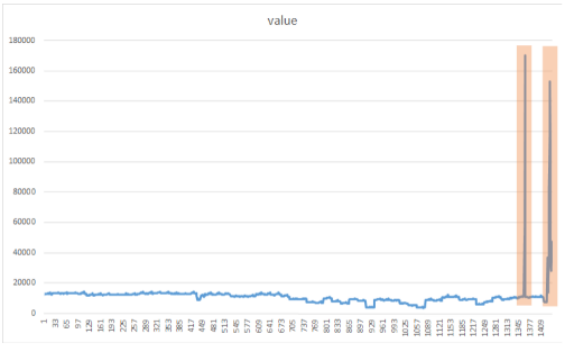


Figura 4.3: Esempio di punti di picco

Lo scopo del rilevamento dei picchi consiste nell'identificare i picchi improvvisi ma temporanei che differiscono notevolmente dalla maggior parte dei valori dei dati delle serie temporali. È importante rilevare questi elementi, osservazioni o eventi rari sospetti in modo tempestivo per fare in modo che abbiano un impatto minimo. La figura 4.3 mostra un esempio di punti di picco.

Change Points

I change points sono modifiche permanenti in una serie temporale di un flusso di distribuzione di valori, come le modifiche di trend. Queste modifiche persistenti durano molto più tempo rispetto agli spikes e potrebbero indicare uno o più eventi catastrofici. I change points non sono in genere visibili a occhio nudo, ma possono essere rilevati nei dati utilizzando un sistema apposito. La figura 4.4 mostra un esempio di punti di modifica.

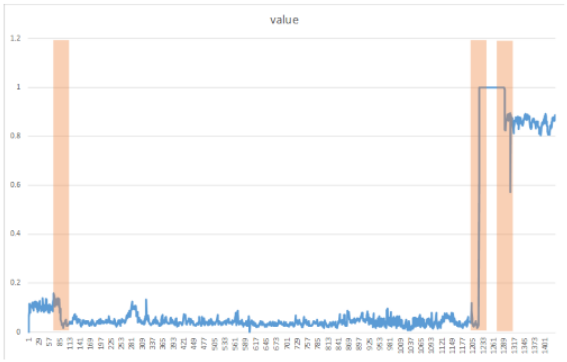


Figura 4.4: Esempio di punti di modifica

4.4.3 Schema della Soluzione

Identificati e definiti cosa siano spike e change points, rimane da spiegare l'ultimo passo ovvero come si è giunti alla conclusione che un device, un allarme e un segnale siano tra loro correlati senza avere una conoscenza pregressa.

La situazione di partenza prevede un input che è stato definito nella sottosezione 4.2.3 ovvero una serie temporale di un segnale, prodotto da un determinato device, sovrapposto ad un allarme che si desidera verificare.

E' bene specificare che il sistema di identificazione di spike e change points permette di inserire un parametro definito come *confidence*, cioè il grado di confidenza con cui un punto di picco o di modifica può essere definito tale.

A questo punto la serie temporale è stata data in input a *ML.NET* con un certo grado di *confidence* e l'output è stato incrociato con i dati degli allarmi.

Quindi, è possibile affermare con un certo grado di *confidence*, che se in un punto in cui si è verificato un allarme sia presente anche uno spike o change point, allora quello può essere un punto significativo in cui si è verificata una rottura o una necessità di manutenzione.

Non avendo a disposizione alcuna informazione dal cliente, queste operazioni sono state necessarie per mettersi in condizione di effettuare operazioni di *PdM*. Ovviamente tutto ciò può essere evitabile nel caso in cui il cliente sia in grado di fornire a priori le situazioni in cui uno o più segnali generino uno o più allarmi.

4.5 Predizione Trend

In questa sezione verrà descritto l'ultimo modulo "collaterale", in cui l'obiettivo è stato quello di utilizzare *Reti Neurali - NN* per predire l'andamento(trend) di una data serie temporale.

Questo può essere utile nel momento in cui si desideri prevedere, con un certo grado di certezza, quale sarà il trend futuro di una serie.

Per sfruttare le *NN*, si è scelto di utilizzare un framework *Python* molto famoso chiamato *Tensorflow*. Il modulo realizzato, per cui, consiste nello sfruttare *Tensorflow* per creare una *NN* di tipo *LSTM*, descritta nella sezione 3.1.6, che sia in grado di predire i valori futuri che assumerà. Il tutto seguito da procedure di validazione e valutazione dell'output.

4.5.1 Tensorflow

TensorFlow è una libreria software open source rivolta alla computazione numerica che utilizza grafi di tipo DataFlow. Il progetto è disponibile sulla piattaforma GitHub ed è stato originariamente sviluppato da ricercatori e ingegneri che lavorano nel team Google Brain nell'ambito dell'organizzazione di ricerca di Google Machine Intelligence per effettuare apprendimento automatico e Deep Learning con reti neurali, ma il sistema è abbastanza generale per essere applicabile in una vasta gamma di altri domini. Il linguaggio di programmazione è Python.

TensorFlow utilizza grafi di tipo DataFlow, dove i nodi del grafo rappresentano operazioni matematiche, mentre gli archi rappresentano array di dati multidimensionali detti tensori che permettono ai nodi di essere collegati: i tensori sono la struttura dati principale di questo tipo di framework.

Questa architettura flessibile permette di sviluppare computazioni su singole o multiple CPUs o GPUs. *TensorFlow* mette a disposizione diversi tipi di API: quelle di più basso livello (*TensorFlow Core*) e alcune di più alto livello che sono costruite sulla base di quelle Core, come, ad esempio, *Keras*.

4.5.2 Keras

Keras è una API di alto livello che permette la costruzione di *NN* ed è utilizzabile da *Tensorflow* e su altri framework. E' stato sviluppato con l'obiettivo di attivare una sperimentazione veloce. *Keras* è una libreria che:

- Permette una prototipazione veloce attraverso la modularità e il fatto che sia estendibile.
- Supporta sia *NN* convoluzionali che ricorrenti.
- Sfrutta ugualmente CPU e GPU.

Keras è l'API che è stata utilizzata per svolgere questo modulo che poi verrà spiegato a livello pratico nel capitolo 5.

4.6 Predizione Remaining Useful Life

Quest'ultimo modulo facente parte del progetto di tesi è quello che più riguarda la *Manutenzione Predittiva - PdM* vero obiettivo iniziale. I moduli precedenti sono tutti serviti da preparazione a questo modulo finale, in quanto non erano stati messi a disposizione ulteriori informazioni sul dataset.

Quando si parla di *PdM*, le possibili attività che si eseguibili sono essenzialmente di due tipologie che rimandano agli utilizzi delle *NN*:

1. **Classificazione Binaria**: fissate n finestre di tempo $w_1, \ldots, w_n$, cercare di capire se si verificherà un guasto durante la finestra n-esima. L'output previsto è di tipo booleano e generalmente in numero delle finestre è due.

2. **Regressione**: si tratta di calcolare il *Remaining Useful Life - RUL*, quindi un numero generalmente intero, che riguarda il tempo rimanente prima che si verifichi un guasto.

All'interno del progetto è stato affrontato solamente il problema dal punto di vista della regressione, ma è già stato impostato se si desidera anche affrontare la classificazione binaria.

La tecnologia utilizzata è sempre il framework *Tensorflow*(sottosezione 4.5.1) in *Python*. Per quanto riguarda invece la gestione del tempo, è pratica sostituire i timestamp con un numero intero che conteggi i cicli da 1 a n, in quanto le *NN* lavorano meglio con riferimenti numerici interi. Ciò è possibile perché i timestamp vengono inseriti nel dataset a intervalli regolari, per cui poi è semplice risalire al tempo originale.

4.6.1 RUL

Il *Remaining Useful Life* è una stima soggettiva del numero di cicli temporali che un componente o un sistema sia in grado di sostenere, svolgendo le sue normali funzioni, prima di avere necessità di manutenzione o prima che si verifichi un guasto.

Un modello di stima per *RUL* non solo fa una previsione, ma fornisce anche un limite di confidenza sulla previsione. Gli input del modello sono features estratte dai dati del sensore il cui comportamento cambia in modo prevedibile quando il sistema si degrada o funziona in diverse modalità.

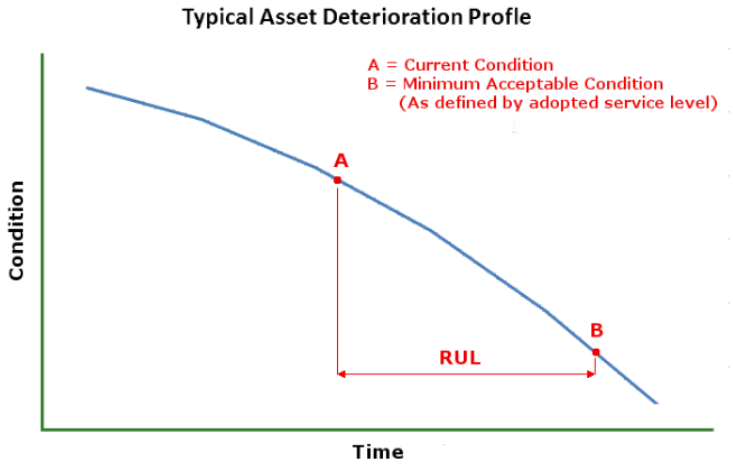


Figura 4.5: Remaining Useful Life

4.6.2 Schema della Predizione

Per sfruttare le *NN* per predire il *RUL* sono necessari alcuni accorgimenti che servono per avere un output che sia il più corretto possibile e che sia comprensibile. Una fase importante è quella del preprocessing che modella il dataset in modo tale che sia un corretto input per la *NN*.

Le fasi di preprocessing sono le seguenti:

1. Selezionare dal dataset iniziale solo le colonne relative al timestamp e al valore.

2. A priori si conosce il valore soglia oltre il quale si verifica un guasto.

3. Si aggiunge una colonna id, in cui si aggiunge un id intero che incrementa di uno ogni volta che si supera la soglia.

4. Dato che i timestamp sono memorizzati costanti, si possono sostituire con numeri interi progressivi che partono da 1 che identificano i cicli.

5. Il ciclo viene ri-azzerato ogni volta che l'id aumenta.

6. Si aggiunge una colonna *RUL* con un numero progressivo pari all'inverso del ciclo; quando il ciclo raggiungerà la soglia il *RUL* sarà zero.

7. Si applica una funzione di normalizzazione alla colonna con il valore e al ciclo per riportarli in un range tra 0 e 1.

L'aspetto del dataset finale è mostrato nella tabella 4.2.

id	**cycle**	**value**	**RUL**	**cycle_norm**
1	1	0.094	616	0.0000
1	2	0.21	615	0.000056
1	3	0.101	614	0.000111

Tabella 4.2: Esempio di dataset normalizzato pronto per *PdM*

L'idea di base consiste nel suddividere la serie temporale in tanti piccole serie determinate dai guasti o dalle manutenzioni. La *NN* per cui ha come obiettivo quello di imparare da queste serie quando avvengono gli interventi per riuscire a predirne il comportamento futuro.

Di seguito la rete viene validata su altre piccole serie temporali dove però non è specificato il momento del guasto: la validazione consiste nel testare la *NN* a indovinare i cicli rimanenti e verificarne poi la correttezza in quanto si conosce questa informazione.

Di particolare importanza è per cui la suddivisione del dataset in train e test set, in cui il numero di id deve essere uguale. Per creare train e test set si suddivide il dataset di partenza in due sulla base della numerazione degli id e la seconda parte viene rinumerata partendo da 1.

La suddivisione ideale tra train e test è 60%-40%, per cui i due gruppi poi vengono rimodellati se necessario arrivare a questa proporzione. La parte di

dataset selezionata come train è quindi pronta per essere data in input mentre la parte di test necessita di una ultima elaborazione.

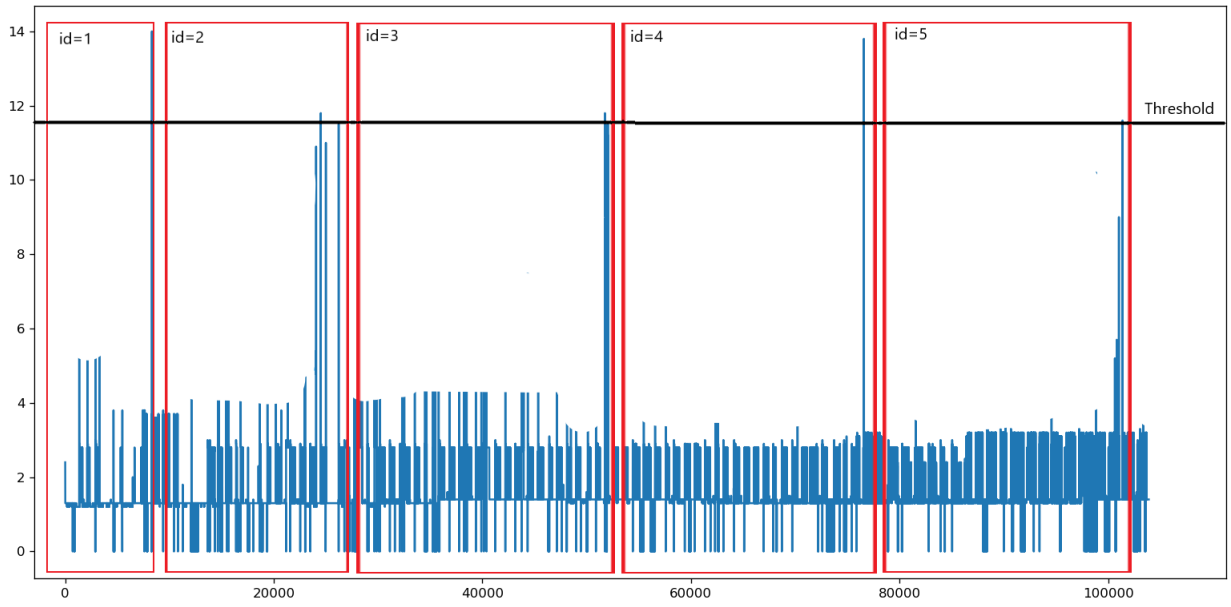


Figura 4.6: Esempio di Training Set

Per ogni id del test set deve essere eliminato un sottogruppo random di righe consecutive partendo sempre dalle ultime e il numero di righe eliminato segnato su un file a parte, questo sarà il *RUL* che la *NN* dovrà predire.

A questo punto la *NN* viene istruita con il train set e validata sul test set verificando sul file salvato il *RUL* predetto.

4.7 Conclusioni

In conclusione, questo capitolo mostra tutto ciò che è stato fatto nel progetto di tesi dal punto di vista della teoria e delle tecnologie applicate. Per i primi quattro moduli non c'è molto altro da aggiungere, l'unica cosa degna

di nota è il fatto che se l'azienda cliente avesse fornito ulteriori informazioni, ciò che è stato fatto avrebbe potuto essere confermato.

Per quanto riguarda invece la parte di predizione del *RUL* c'è da dire che di solito sono necessarie un numero di feature maggiore rispetto a quella unica utilizzata. Inoltre il cliente non ha fornito la soglia entro la quale si sono verificate le manutenzioni, per cui c'è stato un po' di carenza di informazioni necessarie. Tutto è stato inferito e immaginato sulla base della conoscenza sul dataset e sull'esperienza accumulata nel progetto.

In ultimo c'è da segnalare però che la tecnica applicata per il calcolo del *RUL* è corretta e la soluzione è pronta nel caso in cui vengano aggiunte features o ulteriori informazioni utili.

Capitolo 5

Realizzazione del Prototipo

5.1 Introduzione

In questo quinto capitolo verranno presentati tutti i moduli per come sono stati effettivamente realizzati: saranno presenti le parti di codice più rilevanti che mostrano il funzionamento di quello che è il cuore di ogni singolo modulo.

La struttura del sistema in generale prevede un paio di aspetti comuni ad tutti i moduli: la connessione ad database e la visualizzazione. Il sistema è stato creato seguendo la struttura di una applicazione .NET di tipo *Windows Form*, dove la visualizzazione è demandata al *Builder*.

La parte di codice relativa alla visualizzazione verrà omessa in quanto facilmente riproducibile e in gran parte generata dal *Builder*.

La struttura del prototipo segue il paradigma *Model-View-Controller* detto *MVC*, dove la parte di *Model* si occupa della connessione al database, la parte di *View* della visualizzazione e la parte di *Controller* invece si occupa di eseguire le chiamate ai vari moduli.

I moduli realizzati in *Python* vengono attivati dall'applicazione .NET mandando in esecuzione un processo specifico che si occupa solo di eseguire il sorgente.

5.1.1 Connessione al Database

Il database di riferimento di questo progetto è di tipo *MySQL* e ogni modulo lo sfrutta, utilizzando le apposite API, per ricavare le variabili e le informazioni ad esse connesse.

La connessione al database avviene mediante linguaggio *C#* oppure *Python* dove, a livello di codice, sono presenti alcune differenze.

Connessione con C#

La connessione al database della parte di *Model* sfrutta un framework denominato *ADO.NET*. *ADO.NET* è una collezione di classi, interfacce, strutture, e tipi che gestisce l'accesso ai dati all'interno del .NET framework.

Il framework mette a disposizione un *Provider* che si occupa di ricevere le richieste sotto forma di query, interrogare il database e di restituire le risposte, il tutto seguendo una mappatura *ORM* ovvero *Object-Relational-Mapping*. Dal punto di vista dell'applicazione, il *Provider* fornisce oggetti attraverso i quali è possibile aprire una connessione e formulare query.

Il codice 5.1 mostra un esempio di connessione tramite *ADO.NET* per eseguire una query che restituisca tutti i nomi di variabile distinti.

Listing 5.1: Connessione MySQL C#

```
1  public void ShowDistinctVariables(string ConnString,string Factory) {
2        if(Factory.Contains("MySql")) {
3          DbProviderFactory DbFactory =
               DbProviderFactories.GetFactory(Factory);
4          using(DbConnection Conn = DbFactory.CreateConnection()) {
5            try {
6              Conn.ConnectionString = ConnString;
7              Conn.Open();
8              DbCommand Com = Conn.CreateCommand();
9              Com.CommandText = "SELECT DISTINCT VariableName FROM
                   tesispot.variables_values_all";
10
```

```
11            using(DbDataReader Reader = Com.ExecuteReader()) {
12              Counter = 0;
13              while(Reader.Read()) {
14                PrintText(this,Convert.ToString(Counter) + ": " +
                      Reader["VariableName"]);
15                Counter++;
16              }
17              Reader.Close();
18              Conn.Close();
19            }
20          } catch(Exception Ex) {
21            PrintText(this,"[dataReader] Error: " + Ex.Message);
22          } finally {
23            if(Conn.State == ConnectionState.Open) Conn.Close();
24          }
25        }
26      }
27    }
```

Connessione con Python

Anche i moduli scritti in *Python* possiedono codice per comunicare con il database in quanto lo scambio di informazioni con una applicazione .NET non è immediato, per cui il codice 5.2 facilita il tutto connettendosi direttamente. La funzione mostrata svolge la stessa query della 5.1.

Listing 5.2: Connessione MySQL Python

```
1 import mysql.connector
2
3 def connectMySqlDb(user,password,host,db):
4     try:
5         conn =
              mysql.connector.connect(user=user,password=password,host=host,
6                                       database=db)
7     except:
8         print('Connection Error')
```

```
9     return conn
10
11 def getVariablesNames():
12     names = list()
13     try:
14         conn = connectMySqlDb('root','admin','localhost','tesispot')
15         cursor = conn.cursor()
16         query = "SELECT * FROM tesispot.variable_names ORDER BY
               VariableName ASC"
17         cursor.execute(query)
18         result_set = cursor.fetchall()
19
20         for row in result_set:
21             names.append(row[0])
22
23     except mysql.connector.Error as err:
24         print("Error:", err.message)
25         conn.close()
26
27     except:
28         print("Unknown error occurred!")
29         conn.close()
30
31     cursor.close()
32     conn.close()
33     return names
```

5.2 Prototipo Feature Selection e Analisi Correlazione

Il primo prototipo presentato riguarda il modulo di Feature Selection e Analisi di Correlazione. Come descritto nella sezione 4.2, il modulo si sviluppa su vari passi, ma quello più degno di nota riguarda l'analisi delle sequenze di variabili in quanto tutti gli altri sono a lui direttamente collegati.

L'idea di fondo è interrogare il database per ricavare una mappa fatta di una coppia chiave-valore in cui la chiave è formata dalla tupla Alarm-Device e il valore è una lista di Timestamp. Da qui è possibile vedere quante coppie Alarm-Device sono generate. La query che fornisce questa informazione é questa:

```
"SELECT VariableName, VariableDeviceId, Timestamp FROM
    variables_values_all Where VariableName like 'Alarm[%' order by
    VariableName, VariableDeviceId"
```

Partendo dalla mappa così presentata, l'obiettivo ora è, per ogni elemento della mappa, vedere quali variabili sono presenti entro trenta secondi dal Timestamp di ogni allarme.

```
"SELECT Id, Timestamp, TimestampEnd, VariableName, Value,
    VariableDeviceId FROM variables_values_all WHERE Timestamp > " +
    "'" + str(timestamp_start) + "' AND Timestamp <= " + "'" +
    str(timestamp_end) + "' AND VariableDeviceId = " + str(device)"
```

La query mostra come, dato il Timestamp e il Device, venga restituita la lista di variabili richieste. Il codice 5.3 mostra come è stata effettuata questa analisi, ovvero ripetendo la query per ogni Timestamp di ogni coppia nella mappa iniziale. Il risultato è poi salvato in un file.

Listing 5.3: Analisi Sequenze di variabili

```
1
2 for (alarm, num) in alarms_devices_timestamp_dict:
3     result_dict = dict()
4     ids_list = list()
5     for timestamp in alarms_devices_timestamp_dict[(alarm, num)]:
6         results = msc.getVariablesAnalysisDelayed(timestamp -
              datetime.timedelta(seconds=seconds_gap), timestamp, num)
7         for id in results:
8             if not id in ids_list:
9                 ids_list.append(id)
10                current_variable_name = results[id]["VariableName"]
```

```
11                current_value = results[id]["Value"]
12                if current_variable_name in result_dict:
13                    result_dict[current_variable_name] = (current_value
                          + result_dict[current_variable_name][0], 1 +
                          result_dict[current_variable_name][1])
14                else:
15                    result_dict[current_variable_name] = (current_value,
                          1)
```

Questa analisi è fondamentale per capire quali variabili possano essere direttamente collegate ad un allarme, è stata un importate punto di partenza per i moduli successivi.

L'ultimo passo di questo modulo è stato, dato un device, un allarme e un segnale, sovrapporre la serie temporale del segnale agli allarmi generando la situazione descritta dalla figura 4.1. Per raggiungere quel risultato, ciò che è stato fatto è unire i timestamp della serie temporale con quelli dell'allarme in unico dataframe di tre colonne del tipo Timestamp-Alarm-Signal e da questo disegnare il grafo con il codice seguente.

Listing 5.4: Grafo Device-Allarme-Segnale

```
1 sns.set(style="white")
2         fig, ax1 = plt.subplots(figsize=(12,12))
3         ax2 = ax1.twinx()
4         plt.title("Device" + str(device) + alarm_name + serie_name)
5         chart = sns.lineplot(x=result_df.Timestamp,
6                     y=result_df.ValueSignal,
7                     color='blue',
8                     ax=ax1)
9         chart2 = sns.barplot(x=result_df.Timestamp,
10                     y=result_df.ValueAlarm,
11                     color='red',
12                     ax=ax2, linewidth=0.5)
13        chart.set_xticklabels(chart.get_xticklabels(), rotation=45)
14        chart2.set_xticklabels(chart2.get_xticklabels(), rotation=45)
15        change_width(ax2, 0.1)
16        change_width(ax1, 0.1)
```

```
17          plt.subplots_adjust(bottom = 0.2)
18          plt.show()
19          sns.set()
```

5.3 Prototipo Qualità dei Segnali

Il prototipo di questo modulo riguarda essenzialmente un unico script *Python* che si occupa di effettuare una analisi di qualità di un indicatore fornito dall'utente. L'indicatore ovviamente è una delle variabili salvate nel database.

Il codice 5.5 è di facile interpretazione in quanto consiste nell'applicare la media mobile semplice o esponenziale alla serie storica di una variabile ricavata dal database. La finestra scelta per calcolare a ritroso la media è di tre giorni: generalmente nel database gli indicatori che sono stati forniti dal cliente sono salvati sei volte l'ora(la finestra è quindi di 432 step). Sono state omesse due semplici funzioni che si occupano di calcolare e valutare la differenza con la media mobile e fornire un ranking.

Listing 5.5: Analisi Qualità Indicatori

```
1  rows=getRowsByNameAndDevice(variable_selected,device)
2  window=432
3  df = pd.DataFrame(rows, columns = cols)
4  df_selected = pd.DataFrame({'Timestamp': df.Timestamp,'Value':
       df.Value})
5  df_selected.drop_duplicates(subset ="Timestamp", inplace = True)
6  df_selected.Value = pd.to_numeric(df_selected.Value)
7
8  df_selected['SMA'] = df_selected.Value.rolling(window=window+1).mean()
9  df_selected['EMA'] = df_selected.Value.ewm(span=window+1,
       adjust=False).mean()
10 df_selected['DIFF_SMA'] = df_selected.Value - df_selected.SMA
11 df_selected['DIFF_EMA'] = df_selected.Value - df_selected.EMA
12
```

```
13 last_value =
       round(float(df_selected['Value'][df_selected.index[-1]]),2)
14 last_value_sma = round(df_selected['SMA'][df_selected.index[-1]],2)
15 last_value_exp = round(df_selected['EMA'][df_selected.index[-1]],2)
16 diff_sma = round(df_selected['DIFF_SMA'][df_selected.index[-1]],2)
17 diff_exp = round(df_selected['DIFF_EMA'][df_selected.index[-1]],2)
```

5.4 Prototipo Individuazione Punti Significativi

Questo modulo è sicuramente il più articolato in quanto unisce i due linguaggi utilizzati e sfrutta le tecnologie di *Machine Learning* su di cui c'è principale interesse. Per capire al meglio il codice che verrà presentato è opportuno mostrare anche la struttura di *Microsoft ML* e di come lavora quando è alla ricerca di anomalie in serie temporali.

5.4.1 Architettura ML.NET

Un'applicazione ML.NET inizia con un oggetto *MLContext*. Questo oggetto singleton contiene i cosiddetti *Catalog*: un *Catalog* è una factory per il caricamento e salvataggio di dati, le trasformazioni e i componenti di gestione del modello. Ogni oggetto catalogo dispone di metodi per creare diversi tipi di componenti; nel modulo in questione il *Catalog* si chiama *AnomalyDetectionCatalog*.

L'architettura si sviluppa seguendo una pipeline ben definita che porta poi all'esecuzione del training e del modello.

I metodi su cui si fonda il rilevamento di anomalie sono *DetectSpike()* e *DetectChangepoint()*, per comodità verrà mostrato il codice del rilevamento degli spike points che è del tutto analogo al successivo con la differenza che i nomi dei metodi cambiano.

Listing 5.6: Metodo DetectSpike()

```
1  public static List<double> DetectSpike(MLContext MlContext, string
       SpikePath, int DocSize,IDataView DataView,string Varname,
2                        string DeviceNum, string Alarm, int Confidence)
                             {
3        var iidSpikeEstimator =
             MlContext.Transforms.DetectIidSpike(outputColumnName:
             nameof(SeriePrediction.Prediction),
4          inputColumnName: nameof(Serie.ValueSignal),confidence:
               Confidence,pvalueHistoryLength: DocSize / 4);
5        ITransformer iidSpikeTransform =
             iidSpikeEstimator.Fit(CreateEmptyDataView(MlContext));
6        IDataView transformedData =
             iidSpikeTransform.Transform(DataView);
7        var predictions =
             MlContext.Data.CreateEnumerable<SeriePrediction>(transformedData,
8        reuseRowObject: false);
9        var spikeList = new List<double>();
10       using(var file = new StreamWriter(SpikePath + "Device"+
             DeviceNum + Alarm + Varname + ".csv")) {
11         foreach(var p in predictions) {
12           var results =
                 $"{p.Prediction[0]};{p.Prediction[1]:f2};{p.Prediction[2]:F2}";
13           if(p.Prediction[0] == 1) {
14             spikeList.Add(p.Prediction[1]);
15           }
16         }
17       }
18       return spikeList;
19     }
```

Il codice 5.6 mostra bene la pipeline dell'architettura:

1. Inizialmente viene creato l'oggetto *Estimator* che si occupa di eseguire il training del modello.

2. L'oggetto *Trasformer* incapsula il training del modello.

3. Viene eseguita l'operazione *Transform* per effettuare le stime su tutte le righe del modello.

4. Infine viene modellato l'output in una variabile contenente le predizioni che può essere navigata per salvare su file il risultato.

5.4.2 Soluzione Proposta

A questo punto sono stati spiegati tutti gli elementi che compongono il prototipo di questo modulo. Infatti l'idea è stata quella di richiedere all'utente un device, un allarme e un segnale e sulla base di ciò verificare se nella serie storica del segnali ci siano punti di picco o di modifica in corrispondenza di allarmi.

Ricevuto l'input dall'utente il sistema scrive differenti file in cui si trovano le informazioni richieste relativamente alla ricerca di anomalie.

```
1  public Tuple<List<double>,List<double>>
       GenerateAnomaliesDetectionFiles(string SeriePath) {
2        var SpikeList = new List<double>();
3        var ChangePointList = new List<double>();
4        try {
5          int DocSize = File.ReadLines(SeriePath).Count();
6          IDataView DataView =
               MlContext.Data.LoadFromTextFile<Serie>(path:
               SeriePath,hasHeader: true,separatorChar: Separator);
7          SpikeList = DetectAnomalies.DetectSpike(MlContext,SpikePath,
8          DocSize,DataView, Varname,Device,Alarm,Confidence);
9          ChangePointList =
               DetectAnomalies.DetectChangepoint(MlContext,ChangePointPath,
10         DocSize,DataView, Varname,Device,Alarm,Confidence);
11       } catch(FileNotFoundException) { }
12       return Tuple.Create(SpikeList,ChangePointList);
13     }
```

Una altra funzione che attiva un processo per eseguire uno script *Python* si preoccupa di generare il file finale e il grafo.

Ottenute le liste con spike e change points, è possibile ricercare se c'è stato un allarme alla presenza di uno di essi attraverso la funzione *Search-Correlation()*.

L'idea è quella di scorrere il file finale alla ricerca di un timestamp in cui siano presenti sia l'allarme che il segnale e poi verificare se quel valore del segnale sia presente in una delle due liste.

```
1  public List<Tuple<string,string,string,double,string,bool>> Search() {
2      List<Tuple<string,string,string,double,string,bool>> Results =
           new List<Tuple<string,string,string,double,string,bool>>();
3
4      try {
5        using(var reader = new StreamReader(DevicePath + "Device" +
             Device.ToString() + "\\Device" + Device.ToString() + Alarm
             + "Serie" + Serie + ".csv")) {
6          while(!reader.EndOfStream) {
7            var line = reader.ReadLine();
8            var values = line.Split(';');
9            if(values[0] != "Timestamp") {
10             string Timestamp = values[0];
11             double? ValueSignal = null;
12             double? ValueAlarm = null;
13
14             if(values[1] != "") {
15               ValueSignal =
                     double.Parse(values[1].Replace(",","."),CultureInfo.
16               InvariantCulture.NumberFormat);
17             }
18
19             if(values[2] != "") {
20               ValueAlarm =
                     double.Parse(values[2].Replace(",","."),CultureInfo.
21               InvariantCulture.NumberFormat);
22             }
23
24             if(ValueSignal.HasValue && ValueAlarm.HasValue &&
```

```
                SpikeList.Contains(ValueSignal.GetValueOrDefault())) {
25              Results.Add(Tuple.Create(Device,Alarm,Serie,ValueSignal.
26              GetValueOrDefault(),Timestamp,true));
27            }
28
29            if(ValueSignal.HasValue && ValueAlarm.HasValue &&
                  ChangePointList.Contains(ValueSignal.GetValueOrDefault()))
                  {
30              Results.Add(Tuple.Create(Device,Alarm,Serie,ValueSignal.
31              GetValueOrDefault(),Timestamp,false));
32            }
33          }
34        }
35      }
36    } catch(FileNotFoundException) {
37      Results.Add(Tuple.Create("NO CORRELATION
            PRESENT","","",0D,"",true));
38    }
39
40    if(Results.Equals(Tuple.Create("","","",0D,""))) {
41      Results.Add(Tuple.Create("NO CORRELATION
            PRESENT","","",0D,"",true));
42    }
43
44    return Results;
45  }
```

La soluzione dà la possibilità all'utente di poter effettuare questa analisi non solo per un segnale, ma anche per tutti i segnali dati da un determinato device. Per cui all'utente non rimane altro che selezionare solamente un device e un allarme e il sistema esegue queste operazioni descritte per tutti i segnali e ricava quelli che sono correlati(sempre ricordando che c'è un certo grado di confidenza).

```
1 List<string> VarList = new List<string>();
2     foreach(string Signal in Signals) {
3
```

```
4       List<Tuple<string,string,string,double,string,bool>> Results =
            Compute(SpikePath,DevicePath,ChangePointPath,Sep,Confidence,
5       ElementsToShow,"False",Refresh,Signal);
6
7       foreach(Tuple<string,string,string,double,string,bool> Result in
            Results) {
8         if(!Result.Item1.Equals("NO CORRELATION PRESENT") &&
              Result.Item6 && !VarList.Contains(Result.Item3)) {
9           VarList.Add(Result.Item3);
10        }
11      }
12    }
13    TextConsole2.Text = Alarm + " generato da Device " + Device + " e
14    in correlazione con le seguenti variabili: " + Environment.NewLine;
15    foreach(string Var in VarList) {
16      TextConsole2.AppendText(Var + Environment.NewLine);
17    }
```

5.5 Prototipo Predizione Trend

Anche questo prototipo prevede un unico script *Python* che stavolta utilizza le *NN* create con *Tensorflow* e *Keras*(sottosezioni 4.5.1 e 4.5.2).

L'obiettivo, in questo caso, è quello di prevedere come una serie storica si comporterà nel futuro. Si è scelto di utilizzare una rete *LSTM* e di mostrare i 30 step successivi. Uno step è la distanza che c'è tra due Timestamp in una serie storica che può variare a seconda del tipo di variabile (di solito il campionamento è fatto ogni due o ogni dieci minuti).

5.5.1 Import

Gli import dello script sono presentati nel codice 5.7 e mostrano ciò che è necessario per mandare in esecuzione il modulo e mostrare i risultati tramite grafico. La funzione *create_dataset()* serve per preparare train e test set alla

rete neurale trasformandoli in array più consoni alla *NN* e per suddividerli considerando gli step specificati.

Listing 5.7: Import Predizione Trend

```
1 import numpy as np
2 from tensorflow import keras
3 import pandas as pd
4 import seaborn as sns
5 from pylab import rcParams
6 import matplotlib.pyplot as plt
7 from pandas.plotting import register_matplotlib_converters
8 from sklearn.preprocessing import StandardScaler
9
10 def create_dataset(X, y, time_steps=1):
11     Xs, ys = [], []
12     for i in range(len(X) - time_steps):
13         v = X.iloc[i:(i + time_steps)].values
14         Xs.append(v)
15         ys.append(y.iloc[i + time_steps])
16     return np.array(Xs), np.array(ys)
```

5.5.2 Preprocessing

La parte di preproccessing (codice 5.8) mostra come, partendo dal dataframe ricavato dal file con la tabella della serie storica, si suddivida il dataframe stesso in train e test seguendo la proporzione 80-20. In seguito viene applicato uno scaler(StandardScaler) ovvero un strumento che permette di rimodulare i valori tra 0 e 1 e infine creati i dataset attraverso la funzione mostrata nella sottosezione 5.5.1.

Listing 5.8: Preprocessing Predizione Trend

```
1 sns.set(style='whitegrid', palette='muted', font_scale=1.5)
2
3 rcParams['figure.figsize'] = 16, 10
4
```

```
5 TIME_STEPS = 30
6
7 df = pd.read_csv('RTM_3Temperature.csv', parse_dates=['Timestamp'],
      index_col='Timestamp')
8 df = df[['Value']]
9 df.head()
10
11 register_matplotlib_converters()
12 plt.plot(df, label='Value')
13 plt.legend(loc='upper left');
14 plt.show()
15
16 train_size = int(len(df) * 0.8)
17 test_size = len(df) - train_size
18 train, test = df.iloc[0:train_size], df.iloc[train_size:len(df)]
19 print(len(train), len(test))
20
21 scaler = StandardScaler()
22 scaler = scaler.fit(train[['Value']])
23
24 train = train.copy()
25 test = test.copy()
26 train['Value'] = scaler.transform(train[['Value']])
27 test['Value'] = scaler.transform(test[['Value']])
28
29 X_train, y_train = create_dataset(train[['Value']], train.Value,
      TIME_STEPS)
30 X_test, y_test = create_dataset(test[['Value']], test.Value,
      TIME_STEPS)
```

5.5.3 Rete Neurale Keras LSTM

Il codice 5.9 mostra come viene creata e parametrizzata la *NN* con *Keras*: la *NN* è abbastanza classica, il modello è di tipo *Sequential*, ci sono solo due strati e la funzione di loss utilizzata è il *MSE*(sottosezione 3.2.3).

Listing 5.9: Rete Neurale per Predizione Trend

```
model = keras.Sequential()
model.add(keras.layers.LSTM(
  units=128,
  input_shape=(X_train.shape[1], X_train.shape[2])
))
model.add(keras.layers.Dense(units=1))
model.compile(
  loss='mean_squared_error',
  optimizer=keras.optimizers.Adam(0.001)
)
```

5.5.4 Training e Predizione

Il codice 5.10 permette di presentare come sono impostati i parametri per il training della *NN* e come avviene la successiva predizione.

Listing 5.10: Training e Predizione Trend

```
history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=16,
    validation_split=0.1,
    verbose=1,
    shuffle=False)

y_pred = model.predict(X_test)

print('\n# Generate predictions for N samples')
predictions = model.predict(X_test[:TIME_STEPS])
print(scaler.inverse_transform(predictions[:TIME_STEPS]))
```

5.6 Prototipo Predizione RUL

Quest'ultima parte prevede il codice relativo alla vera e propria *PdM*, anch'esso realizzato in *Python* e poi inserito all'interno del sistema .NET. Qui la parte di preproccessing è prevalente e la *NN* è più articolata rispetto alla precedente.

In questa sezione gli import verranno omessi in quanto molto simili ai precedenti.

5.6.1 Preprocessing

Il codice relativo al preproccessing si suddivide in varie parti in quanto più complesso rispetto ai precedenti. Infatti è qui che si spacchetta la serie storica in tante mini serie suddivise da valori soglia e si aggiunge la colonna di *RUL*.

Nel codice 5.11 iniziale viene ricavato il dataframe dal file e da esso inizia tutto il preprocessing: vengono salvati su un altro dataframe i punti di rottura e per ognuno di essi viene creato un id, di seguito viene creato anche il test set.

Listing 5.11: Preprocessing Predizione RUL - Parte 1

```
1  np.random.seed(1234)
2  PYTHONHASHSEED = 0
3
4  model_path = 'C:\\Users\\luca-\\Desktop\\temp\\regression_model.h5'
5
6  df =
       pd.read_csv('C:\\Users\\luca-\\Desktop\\temp\\RTM_1ExhaustCurrentPeak.csv')
7  df = df.rename(columns={"Timestamp": "cycle","Value": "value"})
8  failure_points_df = df.loc[df['value'] >= 11.5]
9  cols = ['id','cycle','value']
10 final_df = pd.DataFrame(columns=cols)
11 training_set = pd.DataFrame(columns=cols)
12 test_set = pd.DataFrame(columns=cols)
13
```

```
ids = 1
start_index = 0
end_index = 0
for index, row in failure_points_df.iterrows():
    end_index = index + 1
    df_append = df.iloc[range(start_index,end_index),:]
    df_append = df_append.copy()
    df_append['id'] = ids
    df_append['cycle'] = range(1, len(df_append) + 1)
    df_append = df_append.loc[:, cols]
    final_df = final_df.append(df_append, ignore_index=True)
    start_index = end_index
    ids = ids + 1

final_df = pd.read_csv('C:\\Users\\luca-\\Desktop\\f.csv')
cols = ['id','cycle','value']
training_set = final_df.loc[final_df['id'] > 20]
training_set.id = training_set.id - 20
training_set = training_set.reset_index(drop=True)
test_set = final_df.loc[final_df['id'] <= 20]
test_set = test_set.reset_index(drop=True)
```

La seconda parte (codice 5.12) fa vedere come ai dataframe di train e test vengano aggiunte le colonne di RUL e le finestre w_0 e w_1 che servono nel caso poi si vorrà fare classificazione binaria. Infine viene creato il dataframe chiamato *truth* che contiene una colonna con il numero di cicli rimanenti per ogni id nel test set.

Listing 5.12: Preprocessing Predizione RUL - Parte 2

```
truth_df_final = pd.DataFrame(columns=['0'])
test_set_final = pd.DataFrame(columns=cols)

for i in range(1,21):
    tmp_df = test_set.loc[test_set['id'] == i]
    num = r.randrange(2,len(tmp_df))
    tmp_df2 = tmp_df.iloc[range(0,num),:]
    truth_df_final = truth_df_final.append({'0': len(tmp_df)-num},
```

```
        ignore_index=True)
    test_set_final = test_set_final.append(tmp_df2, ignore_index=True)

rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)

w1 = 30
w0 = 15
train_df['label1'] = np.where(train_df['RUL'] <= w1, 1, 0 )
train_df['label2'] = train_df['label1']
train_df.loc[train_df['RUL'] <= w0, 'label2'] = 2

rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)

test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)

test_df['label1'] = np.where(test_df['RUL'] <= w1, 1, 0 )
test_df['label2'] = test_df['label1']
test_df.loc[test_df['RUL'] <= w0, 'label2'] = 2
```

L'ultima parte di preprocessing(codice 5.13) applica uno scaler al dataframe di train e test.

Listing 5.13: Preprocessing Predizione RUL - Parte 3

```
train_df['cycle_norm'] = train_df['cycle']
cols_normalize =
    train_df.columns.difference(['id','cycle','RUL','label1','label2'])
```

```
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df =
    pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                           columns=cols_normalize,
                           index=train_df.index)
join_df =
    train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)

test_df['cycle_norm'] = test_df['cycle']
norm_test_df =
    pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                          columns=cols_normalize,
                          index=test_df.index)
test_join_df =
    test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
```

5.6.2 Preparazione Dataframe

Sistemati i dataframe di train, test e truth è necessario prepararli alla *NN*: nel codice 5.14 vengono mostrate le due funzioni che fanno ciò(*gen_sequence* e *gen_labels*). La funzione *r2_keras* permette di applicare l metrica R^2 alla regressione che viene fatta.

Listing 5.14: Preparazione Dataframe RUL

```
def gen_sequence(id_df, seq_length, seq_cols):
    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]
    for start, stop in zip(range(0, num_elements-seq_length),
        range(seq_length, num_elements)):
        yield data_matrix[start:stop, :]

def gen_labels(id_df, seq_length, label):
    data_matrix = id_df[label].values
```

```
    num_elements = data_matrix.shape[0]
    return data_matrix[seq_length:num_elements, :]

def r2_keras(y_true, y_pred):
    SS_res = K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )

sequence_cols = ['value', 'cycle_norm']

l=list()
for id in train_df['id'].unique():
    l.append(len(train_df[train_df['id']==id]))

sequence_length = min(l) - 1

seq_gen = (list(gen_sequence(train_df[train_df['id']==id],
    sequence_length, sequence_cols))
          for id in train_df['id'].unique())

seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)

label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length,
    ['RUL'])
            for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
label_array.shape

nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]
```

5.6.3 Rete Neurale, Training e Validazione

Il codice 5.15 espone il modello utilizzato, che si differenzia dalla sezione precedente in quanto sono utilizzati più strati. La parte di training utilizza i parametri mostrati. La validazione è classica e sfrutta la metrica.

Listing 5.15: Preparazione Dataframe RUL

```
model = Sequential()
model.add(LSTM(
         input_shape=(sequence_length, nb_features),
         units=100,
         return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=nb_out))
model.add(Activation("linear"))
model.compile(loss='mean_squared_error',
    optimizer='rmsprop',metrics=['mae',r2_keras])

print(model.summary())

history = model.fit(seq_array, label_array, epochs=100, batch_size=32,
                    validation_split=0.1, verbose=1)
scores = model.evaluate(seq_array, label_array, verbose=1,
    batch_size=200)
print('\nMAE: {}'.format(scores[1]))
print('\nR^2: {}'.format(scores[2]))

y_pred = model.predict(seq_array,verbose=1, batch_size=200)
y_true = label_array

test_set = pd.DataFrame(y_pred)
```

Capitolo 6

Risultati

6.1 Introduzione

Quest'ultimo capitolo riguardante il caso di studio si occuperà di mostrare i grafici e i risultati di ciò che è stato affrontato fin'ora. Il caso di studio ben si è prestato per testare le moderne tecnologie di *Machine Learning* e di *Manutenzione Predittiva* anche se un minimo più di informazioni da parte del cliente avrebbero permesso perlomeno di validare e capire se ciò che è stato fatto fosse perlopiù corretto e dove apportare cambiamenti.

I risultati verranno presentati accorpando i moduli simili, seguendo comunque la struttura generale dei capitoli precedenti.

6.2 Risultati Primi Due Moduli

I primi due moduli sono quelli riguardanti la *Feature Selection* e *Analisi di Correlazione* e la *Index Quality*: di seguito verranno mostrati alcuni esempi di risultati ottenuti per alcune tipologie di variabili, data la quantità di variabili e possibili combinazioni disponibili.

6.2.1 Feature Selection

I risultati relativi a questo modulo riguardano fondamentalmente l'analisi degli allarmi presentato nel codice 5.3: la tabella 6.1 fa vedere giusto un frammento di quello che è il risultato dell'analisi in quanto la tabella vera e propria avrebbe circa 13 mila righe.

Alarm	**Device**	**VariableName**	**Value(Avg)**	**Occurency**	**Gap(Sec)**
Alarm[0]	14	RTM_MachineStatus	1,56	9	30
Alarm[0]	14	STA_204LowPowerTime	86,46	13	30
Alarm[0]	14	STA_204PowerTime	572,62	13	30
Alarm[0]	14	STA_254LowPowerTime	135,92	13	30
Alarm[0]	14	STA_254PowerTime	7000,23	13	30
Alarm[0]	14	STA_304LowPowerTime	0,00	13	30
Alarm[0]	14	STA_304PowerTime	0,00	13	30
Alarm[0]	14	STA_354LowPowerTime	0,00	13	30
Alarm[0]	14	STA_354PowerTime	0,00	13	30
Alarm[0]	14	STA_404LowPowerTime	0,00	13	30
Alarm[0]	14	STA_404PowerTime	0,00	13	30
Alarm[0]	14	STA_454LowPowerTime	0,00	13	30

Tabella 6.1: Analisi di Alarm[0] e Device 14

Da questo esempio è possibile capire quale sequenza si verifichi quando l'allarme scaturisce da un determinato device, viene inoltre fornita una media dei valore del segnale durante la sequenza e il numero di volte che occorre.

Dalla tabella che scaturisce da questa analisi, è possibile raggruppare le sequenze di variabili dati i singoli device ed ottenere una tabella come la 6.2(ovviamente quella mostrata è solo una piccola parte).

Oltre alla sequenza di variabili, la tabella 6.2 affianca la media ottenuta con l'analisi precedente con la media originale della variabile e affianca anche la differenza che c'è.

L'analisi di correlazione invece ha portato risultati già mostrati nelle figure 2.4 e 4.1.

Device	**SequenzeDiVar**	**Avg (30 Sec)**	**Avg Tabella Originale**	**Diff**
6	STA_PowerOnTime	482.615,33	398.532,80	84.082,53
6	STA_AutomaticTime	473.079,67	453.438,46	19.641,21
6	STA_SprayingTime	495.019,75	461.526,78	33.492,97
6	STA_ConveyorMeters	1.986.830,75	1.642.990,73	343.840,02
6	STA_XAxisMeters	37.100.375,25	35.386.691,79	1.713.683,46
6	STA_PaintedSurface	1.094.291,50	749.931,60	344.359,90
6	STA_PaintedPieces	3.183.790,75	2.203.549,54	980.241,21
6	STA_DoorOpen	153,00	485,93	-332,93

Tabella 6.2: Parte delle sequenze di variabili collegabili al device 6

La tabella 6.3 mostra delle statistiche di base riguardanti le variabili che sono state utilizzate per le varie computazioni nei moduli successivi.

Le quattro variabili presentate sono state scelte per determinate caratteristiche che possiedono, a parte quella utilizzata per il ranking degli indici che è stata direttamente indicata dal cliente.

Per l'analisi delle anomalie, la scelta è ricaduta su tipologie di serie storiche che avessero trend non regolare con picchi positivi o negativi che potessero indicare una eventuale anomalia.

La variabile *RTM_3Temperature* è stata scelta invece per la predizione in quanto è quella che possiede più occorrenze, quindi più adatta per una *NN*.

VariableName	**num**	**avg**	**median**	**min**	**max**	**stdev**
RTM_3Temperature	613.921	38,90	39,40	10,00	54,80	6
STA_PaintedSurface	46.060	749.931,60	924.882,50	0,00	1.254.634,00	469.406
PNL_ThermalPowerProdPeak	17.806	850,50	880,53	0,00	1.839,95	247
RTM_BoilerInputTemperatureMean	17.799	86,31	86,40	76,20	93,90	2

Tabella 6.3: Statistiche variabili significative considerate

La tabella 6.4 fa vedere la sequenza di variabili relative a *Device 1-Alarm[390]* e *RTM_BoilerInputTemperatureMean* . Sono state scelte questi parametri in quanto essa è una serie storica che è stata utilizzata anche nel

modulo successivo per l'analisi delle anomalie e, data la sua conformazione, ben si presta a questo tipo di analisi.

Alarm	Device	VariableName	Value(Avg)	Occurency	Gap(Sec)
Alarm[390]	1	RTM_ThermalPowerProductMean	640,04	523	30
Alarm[390]	1	RTM_ThermalPowerProductPeak	646,03	524	30
Alarm[390]	1	RTM_ThermalPowerProductMin	132,49	522	30
Alarm[390]	1	RTM_SmokeTempMotorOutMean	478,30	521	30
Alarm[390]	1	RTM_SmokeTempMotorOutPeak	478,60	522	30
Alarm[390]	1	RTM_SmokeTempMotorOutMin	54,02	523	30
Alarm[390]	1	RTM_SmokeTempCatalystOutMean	482,99	523	30
Alarm[390]	1	RTM_SmokeTempCatalystOutPeak	483,22	525	30
Alarm[390]	1	RTM_SmokeTempCatalystOutMin	56,91	526	30
Alarm[390]	1	RTM_SmokeTempChimneyOutMean	137,39	527	30
Alarm[390]	1	RTM_SmokeTempChimneyOutPeak	138,93	527	30
Alarm[390]	1	RTM_SmokeTempChimneyOutMin	15,26	527	30
Alarm[390]	**1**	**RTM_BoilerInputTemperatureMean**	**85,52**	**527**	**30**
Alarm[390]	1	RTM_BoilerInputTemperaturePeak	86,14	527	30

Tabella 6.4: Analisi sequenze di *RTM_BoilerInputTemperatureMean*

Alarm	Device	VariableName	Value(Avg)	Occurency	Gap(Sec)
Alarm[997]	1	PNL_ThermalPowerProdMean	597,02	6	30
Alarm[997]	**1**	**PNL_ThermalPowerProdPeak**	**685,35**	**6**	**30**
Alarm[997]	1	PNL_ThermalPowerProdMin	564,80	6	30
Alarm[997]	1	PNL_ExaustSmokeMean	473,27	3	30
Alarm[997]	1	PNL_ExaustSmokePeak	320,37	3	30
Alarm[997]	1	PNL_ExaustSmokeMin	319,93	3	30
Alarm[997]	1	PNL_ActiveElPowerMean	0,00	6	30
Alarm[997]	1	PNL_ActiveElPowerPeak	0,00	6	30
Alarm[997]	1	PNL_ActiveElPowerMin	0,00	6	30
Alarm[997]	1	PNL_ReleasedCH4PowerMean	2.524,97	6	30
Alarm[997]	1	PNL_ReleasedCH4PowerPeak	2.084,77	6	30

Tabella 6.5: Analisi sequenze di *PNL_ThermalPowerProdPeak*

La tabella 6.4 presenta la sequenza relativa alla variabile, la differenza con la media originale non è significativa, ma l'analisi di anomalie poi presenterà lo stesso punti interessanti.

La tabella 6.5, invece, fa vedere la sequenza di *PNL_ThermalPowerProdPeak* scaturita dall'analisi effettuata come sulle altre variabili, qui la differenza con la media iniziale è più evidente e più significativa.

6.2.2 Index Quality

L'analisi di qualità degli indicatori, che ha avuto uno sviluppo abbastanza lineare, ha portato a grafici della tipologia della figura 6.2.

variable_name=STA_PaintedSurface
mean=749931.6010362571
variance=220341684363.98184
covariance=220346468264.7258
Last Value -> 1254634.0
Simple Moving Average (Window=72hour) -> 1251284.31
Exponential Moving Average (Window=72hour) -> 1251138.22
Diff SMA -> 3349.69
Diff EMA -> 3495.78
SMA_DIFF -> 0.267%
EMA_DIFF -> 0.2786%
RANK SMA -> 1°
RANK EMA -> 1°

Figura 6.1: Statistiche e Ranking su *STA_PaintedSurface*

La figura 6.2 fa vedere uno degli indici indicati dal cliente *STA_PaintedSurface* che è presente tra le variabili e che indica la superficie che è stata verniciata. L'andamento della *SMA* e della *EMA* è lineare in quanto non sono presenti grandi cambiamenti di trend.

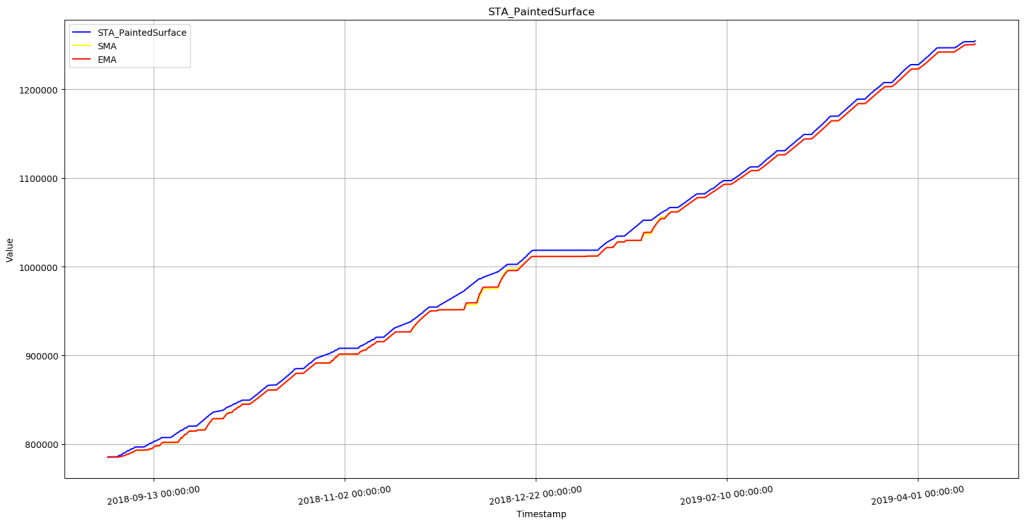


Figura 6.2: SMA e EMA su *STA_PaintedSurface*

6.3 Risultati Individuazione Punti Significativi

Il modulo che riguarda il rilevamento delle anomalie mostrato nella sezione 5.4 ha portato i risultati più soddisfacenti. Infatti, partendo solamente da un dataset di variabili e nulla di più, si è riuscito a stabilire se ci fosse una possibile correlazione tra di esse.

Gli esempi che verranno mostrati nelle figure successive mostreranno un estratto di ciò che è possibile fare con il sistema creato: in questi esempi sono stati presi solamente due device, due allarmi e una variabile che sembrano essere più significativi e più adatti ad essere presentati in quanto la serie storica presenta picchi ad occhio nudo e gli allarmi sono molto frequenti.

Per il rilevamento dei punti di picco e di modifica la variabile scelta è stata *RTM_BoilerInputTemperatureMean*, la quale ha una serie storica che varia abbastanza nel tempo. Il sistema ha riconosciuto tutto i possibile punti di picco e li ha confrontati l'allarme scelto, che in questo caso era *Alarm[390]*, tutto generato dal device 1. Il risultato è mostrato nelle figure seguenti:

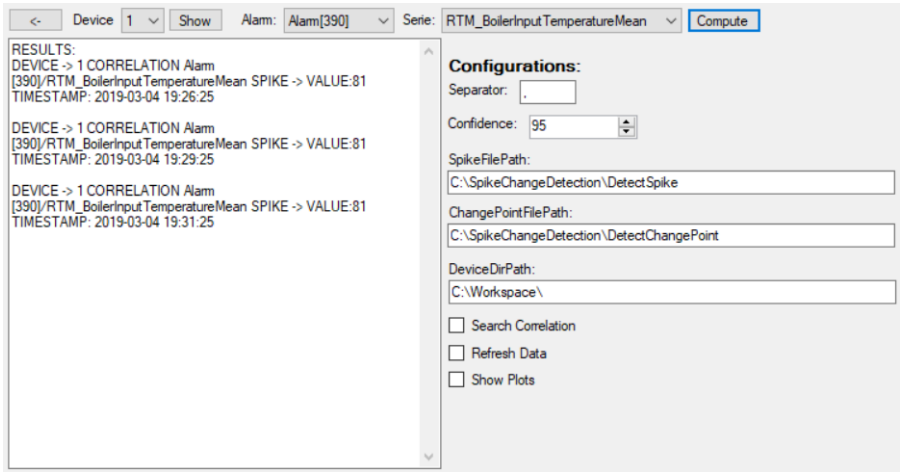


Figura 6.3: Risultato della correlazione tra *RTM_BoilerInputTemperatureMean* e *Alarm[390]* da *Device 1*

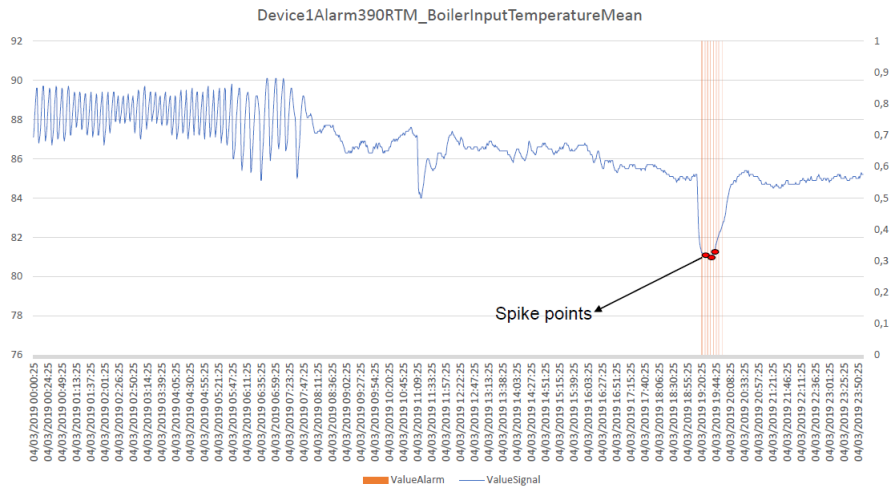


Figura 6.4: Grafico di *RTM_BoilerInputTemperatureMean* con evidenziati i punti di picco in corrispondenza di allarmi

Come ultima parte di questo modulo verrà mostrato l'output finale della esecuzione del processo di ricerca di anomalie associate ad allarmi eseguito per tutte le variabili scaturite da un device e confrontate con un allarme. In questo caso è stato scelto il device 16 e *Alarm[1]*.

L'analisi di correlazione finale ha evidenziato che i segnali correlati sono quelli mostrati nella figura seguente.

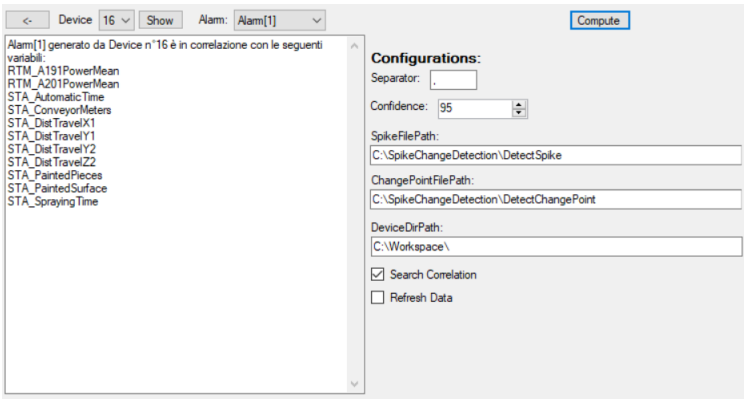


Figura 6.5: Output dell'analisi su tutte le variabili generate dal *Device 16* confrontate con *Alarm[1]*

Per quanto riguarda invece il secondo segnale preso in esame, *PNL_ThermalPowerProdPeak*, l'analisi è stata condotta incrociando i dati con quelli relativi a *Alarm[997]* sempre mantenendo il *Device 1*.

I risultati che ne sono conseguiti sono mostrati dalle due figure seguenti.

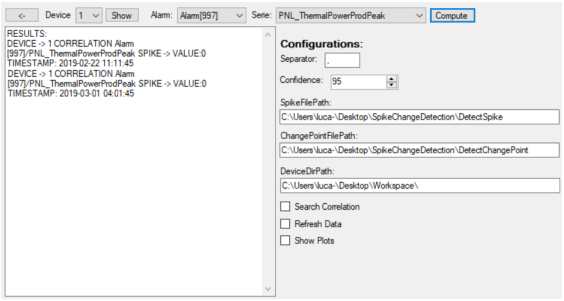


Figura 6.6: Risultato della correlazione tra *PNL_ThermalPowerProdPeak* e *Alarm[997]* da *Device 1*

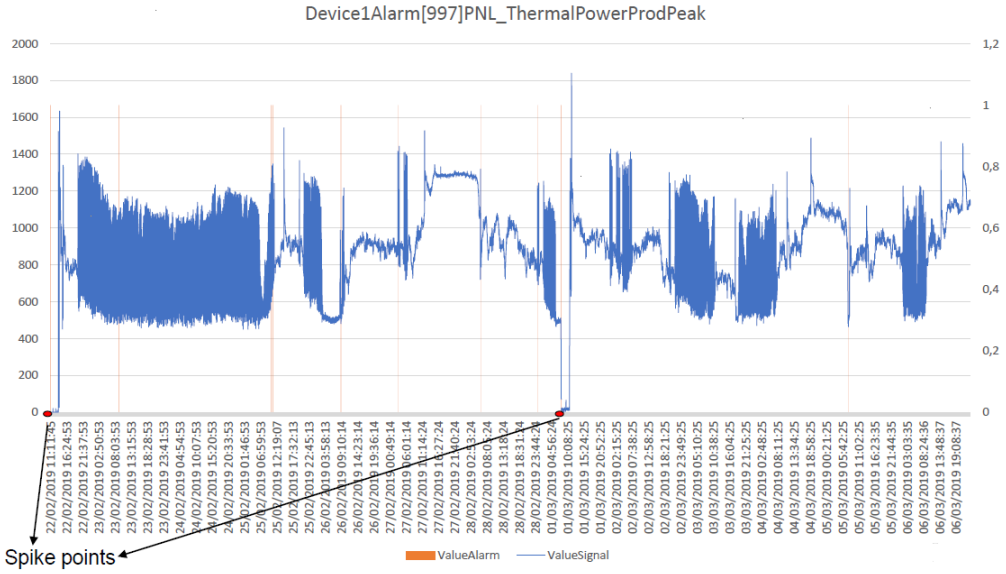


Figura 6.7: Grafico di *PNL_ThermalPowerProdPeak* con evidenziati i punti di picco in corrispondenza di allarmi

6.4 Risultati Predizione

Ciò che manca da presentare sono gli ultimi risultati relativi alla predizione con *NN*: *Predizione Trend* e *Predizione RUL*. Entrambi, come visto già nelle sezioni 4.5 e 4.6, sfruttano le *NN* per imparare dai dati già esistenti.

Con tutte e due le casistiche ci si trova nell'ambito della regressione in quanto entrambi moduli mirano a prevedere un valore: nel primo caso si tratta di un valore di una serie storica, mentre nel secondo il valore del *RUL*.

6.4.1 Risultati Predizione Trend

Questo è il modulo che riguarda la predizione di una serie storica sulla base dei suoi dati precedenti. Come già detto, questo è un modulo più collaterale, ma comunque utile per avere una idea generale dell'andamento futuro e di quanto sia corretta la *NN*.

La *NN* è stata impostata con i parametri indicati nel codice 5.10 e la serie storica scelta è la variabile denominata *RTM_3Temperature*, la quale è anche quella che presenta più occorrenze(circa 600 mila). La predizione è stata effettuata sulle successive 30 occorrenze.

La figura 6.8 mostra la serie storica in verde, in blu la parte di serie utilizzata per il testing e in rosso la predizione di test, in giallo è presente la predizione futura.

L'asse x non presenta i timestamp veri e propri perché sono stati sostituiti da time steps in quanto le occorrenze vengono salvate nel database sempre ciclicamente.

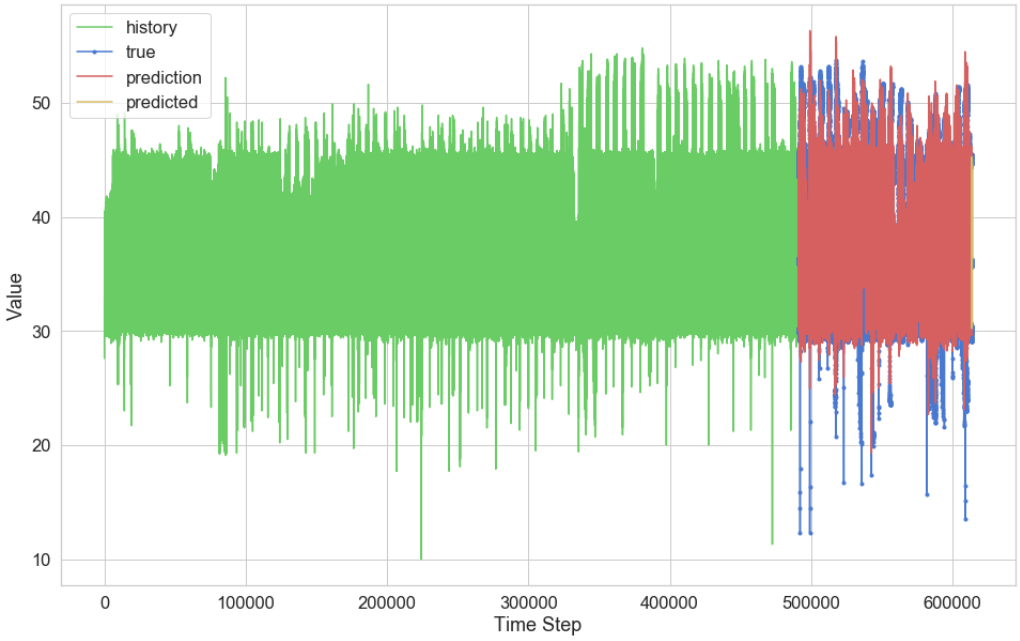


Figura 6.8: Train, Test e Predizione di *RTM_3Temperature*

La figura 6.9 invece riguarda la funzione di loss applicata, ovvero la *MSE*. Il risultato è abbastanza buono, soprattutto in fase di training, dove il valore di loss è pressoché vicino allo zero.

Per quanto riguarda la fase di test, il valore di loss a cui si giunge è circa 0.2, che è un buon valore.

La stessa *NN* è stata poi testata su una serie storica con meno occorrenze, cioè quella relativa alla variabile *RTM_1ExhaustCurrentPeak*(circa 100 mila).

Il risultato che ne è conseguito è molto simile, ma con valori di loss leggermente maggiori in quanto le occorrenze potevano essere un numero maggiore.

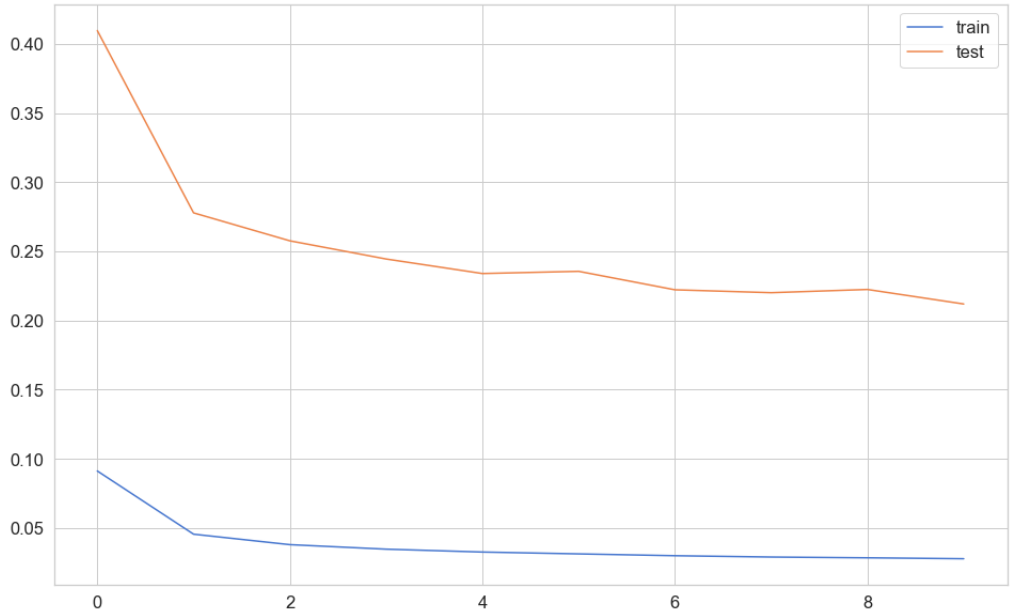


Figura 6.9: Funzione di Loss di *RTM_3Temperature*

6.4.2 Commento Predizione RUL

Dalla parte di predizione del *RUL* non è stato possibile ricavare risultati, in quanto i dati così come sono stati ottenuti, non sono stati sufficienti.

Ciò che era fondamentale conoscere in questo modulo era come le variabili fossero in correlazione l'uno con l'altro a prescindere dai device e sopratutto quando i casi in cui si sono presentate situazioni di guasto o manutenzione.

Tutti questi dati il cliente non è stato ancora in grado di fornire, per cui questo modulo è completo nella parte di codice, ma non nei risultati.

Eseguire una simulazione con dati fittizi non è sembrata una buona alternativa perché erano troppe le informazioni simulate e quindi troppo veicolate.

Capitolo 7

Conclusioni e Sviluppi Futuri

In questa tesi è stato presentato un progetto proposto da una azienda che produce software (*SPOT*) per applicare tecniche di *Machine Learning* e *Manutenzione Predittiva* su dati provenienti da una azienda cliente(*Cefla Finishing*) che lavora in ambito manifatturiero.

La situazione di partenza, per cui, prevedeva un database con dati provenienti da macchine che si occupano di stampaggio e verniciatura su differenti materiali. A questi dati non era stata però aggiunta alcuna ulteriore informazione, perciò i risultati che ne sono conseguiti sono stati totalmente inferiti dai dati di partenza.

I moduli realizzati sono stati cinque, mostrati approfonditamente nei capitoli precedenti. All'azienda cliente sono stati presentati i primi moduli relativi all'*Analisi di Correlazione* tra variabili e alla *Feature Selection*, i quali hanno riscosso successo.

Il componente del sistema che si occupa di *Index Quality* è stato espressamente richiesto dal cliente, mentre, per i componenti finali, non c'è stata occasione per una presentazione.

Per quanto riguarda il modulo di predizione del *Remaining Useful Life*, esso non è completo di risultati perché le informazioni fornite non sono state sufficienti. L'azienda cliente è di grandi dimensioni, le comunicazioni con essa non sono sempre facili e immediate; per questi motivi non si è potuto

raggiungere un risultato, ma si è potuto solamente impostare il lavoro.

Per quanto riguarda lo sviluppo futuro, il punto di partenza dovrebbe essere quello di verificare, prima di tutto, se le correlazioni trovate (device-allarme-segnale) siano effettivamente veritiere.

Il passo successivo sarebbe poi raccogliere i dati relativi alle manutenzioni e ai guasti per capire cosa succede in concomitanza di ciò e quali variabili siano effettivamente più rilevanti.

Così facendo, si potrebbe applicare il modulo di predizione del *RUL*, sfruttando tutte le nuove informazioni che sarebbero utili al conseguimento del risultato.

In conclusione, ciò che manca sono essenzialmente dei dati che devono essere ottenuti dialogando con il cliente oppure anche attraverso interviste con operatori che lavorano sulle macchine e che hanno conoscenza degli impianti.

Questo progetto di tesi fornisce una base solida e mostra come improntare un progetto che utilizzi *Manutenzione Predittiva* partendo solamente da un database di allarmi e segnali senza informazioni ulteriori.

Bibliografia

[1] S. Diamond and A. Marfatia, 2013. *"Predictive maintenance for dummies"*, Hoboken.

[2] Stephan Spiegel, Fabian Mueller, Dorothea Weismann, and John Bird, 2018. *"Cost-sensitive learning for predictive maintenance".*

[3] McKinsey Report, 2015. *"How to navigate digitization of the manufacturing sector".*

[4] Jon Bokrantz, Anders Skoogh, Cecilia Berlin, and Johan Stahre, 2017. *"Maintenance in digitalised manufacturing: Delphi-based scenarios for 2030"*, International Journal of Production Economics.

[5] Xiaoning Jin, Brian A. Weiss, David Siegel, and Jay Lee, 2016. *"Present status and future growth of advanced maintenance technology and strategy in us manufacturing"*, International journal of prognostics and health management.

[6] Vincent F. A. Meyer zu Wickern, 2019. *"Challenges and Reliability of Predictive Maintenance"*. Rhein-Waal University Of Applied Sciences, Faculty of Communication and Environment.

[7] Kevin A. Kaiser and Nagi Z. Gebraeel, 2009. *"Predictive maintenance management using sensor-based degradation models".*

[8] K. Efthymiou, N. Papakostas, D. Mourtzis, and G. Chryssolouris, 2012. *"On a predictive maintenance platform for production systems".*

[9] Morteza Soltani, 2018. *"Joint optimization of opportunistic predictive maintenance and multi-location spare part inventories for a deteriorating system considering imperfect actions"*.

[10] Nagi Z. Gebraeel, Mark A. Lawley, Rong Li, and Jennifer K. Ryan, 2005. *"Residual-life distributions from component degradation signals: A bayesian approach"*.

[11] J. Venkatesh, 2007. *"An introduction to total productive maintenance (tpm)"*.

[12] Seyed Iravani and Izak Duenyas, 2002. *"Integrated maintenance and production control of a deteriorating production system"*.

[13] Jay Lee, Hung-An Kao, and Shanhu Yang, 2014. *"Service innovation and smart analytics for industry 4.0 and big data environment"*.

[14] Bernard Marr, Forbes, 2016. *"What Is The Difference Between Artificial Intelligence And Machine Learning?"*.

[15] Sepp Hochreiter, Jurgen Schmidhuber, 1997. *"Long-Short Term Memory"*.

[16] Ricardo Buettner, 2016. *"Predicting user behavior in electronic markets based on personality-mining in large online social networks: A personality-based product recommender framework"*.

[17] Charles Nyce, 2007. *" Predictive Analytics White Paper"*.

[18] Wayne Eckerson, 2007. *"Extending the Value of Your Data Warehousing Investment"*.

Ringraziamenti

Desidero innanzitutto ringraziare i prof. Vittorio Maniezzo e Antonella Carbonaro per la professionalità, attenzione, disponibilità e pazienza che mi hanno dimostrato durante tutto il percorso di tesi.

Desidero ringraziare l'ing. Maurizio Pensato, che con la sua azienda *SPOT Software* mi ha dato la possibilità di misurarmi con un caso reale e con tutte le implicazioni che ne conseguono e di lavorare per una grande azienda come *Cefla*, che ringrazio.

Desidero poi ringraziare tutta la mia famiglia, dai miei genitori a mia sorella, ai miei nonni e ai miei zii, che ogni giorno mi ha sopportato e supportato.

Desidero ringraziare la mia fidanzata Federica che più di tutti ha saputo capirmi e sopportarmi.

Per ultimi ma non meno importanti ringrazio tutti i miei amici, quelli storici, quelli che hanno condiviso con me il percorso dell'università e quelli che ogni giorno condividono con me la passione per la pallacanestro.