Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica, Artificial Intelligence and Data Analytics

A.a. 2024/2025

Sessione di laurea Luglio 2025

Piattaforma di Manutenzione Predittiva per Flotte di Veicoli

Relatore: Paolo Garza

Candidato: Matteo Bollo

Ai miei genitori,
stelle fisse nel mio cielo,
luci di speranza e custodi del mio cammino.
Con il vostro amore silenzioso e la vostra forza invisibile,
avete nutrito i miei sogni e dato ali ai miei desideri.
Senza di voi, questo viaggio non avrebbe avuto inizio,
né alcun traguardo avrebbe avuto senso.
Grazie per essere il mio rifugio e la mia guida,
per aver trasformato ogni ostacolo in un passo verso la luce.

Sommario

In questa tesi viene esplorata l'applicazione della manutenzione predittiva per l'ottimizzazione della gestione delle flotte di veicoli, sfruttando tecniche di Machine Learning e Big Data. L'obiettivo principale del caso di studio è lo sviluppo di un sistema in grado di predire la necessità di intervenire su una determinata parte meccanica prima che questa si guasti, riducendo i tempi di fermo macchina e ottimizzando i costi. Il progetto si basa su un modello predittivo costruito utilizzando dati storici e telemetrici combinati provenienti dai veicoli stessi. Dopo un'analisi dei possibili algoritmi da utilizzare, è stato adottato un approccio di stacking, combinando 5 modelli per migliorare la precisione delle previsioni. Le prestazioni del sistema sono state valutate secondo due metriche: Root Mean Squared Error (RMSE) e Mean Absolute Error (MAE), evidenziando una buona accuratezza nelle previsioni, sopratutto per le componenti soggette ad usura graduale. Dal punto di vista tecnico, il sistema è stato sviluppato utilizzando Apache Spark per l'elaborazione dei dati, un'API Flask Python per l'esposizione del modello predittivo, un server Nest.js che comunica ed elabora i dati da passare alla UI, la quale è sviluppata usando Vue.js combinato a PrimeVue. L'intero sistema è stato poi suddiviso in 3 container Docker separati e distribuito su Google Cloud Run, garantendo così scalabilità e accessibilità. Per migliorare l'esperienza utente e l'efficienza dell'interfaccia grafica, è stato introdotto un sistema di caching, basato su localStorage riducendo il carico su server e su API. I risultati ottenuti dall'analisi, dimostrano che l'uso della manutenzione predittiva può portare a diversi vantaggi operativi, tra cui riduzione dei costi di intervento, una maggiore affidabilità dei veicoli e una migliore ottimizzazione delle risorse a disposizione. Tuttavia, il lavoro ha incontrato alcune limitazioni e punti critici, come la difficoltà nel prevedere guasti improvvisi e la necessità di integrare dati ambientali per migliorare la predizione. Questa tesi, fornisce un contributo nel campo della manutenzione predittiva, proponendo alle aziende un sistema innovativo e scalabile, con il potenziale di migliorare l'efficienza operativa e di riduzione dei costi di gestione delle flotte.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Contesto e Motivazioni

Negli ultimi anni, l'industria automobilistica ha subito una trasformazione significativa grazie all'evoluzione delle tecnologie digitali e all'adozione di servizi innovativi, tra cui la manutenzione predittiva. L'integrazione di sistemi avanzati per l'analisi dei dati e l'intelligenza artificiale ha consentito alle aziende di ottimizzare le strategie di gestione della manutenzione delle flotte di veicoli, apportando numerosi benefici in termini di efficienza operativa e riduzione dei costi. La manutenzione dei veicoli, ha da sempre seguito due approcci principali:

- **Manutenzione correttiva**: prevede l'intervento solo dopo che si verifica un guasto o un malfunzionamento. Questo metodo, è molto semplice da implementare, ma presenta anche molti svantaggi, tra cui tempi di fermo imprevisti, costi elevati di riparazione e possibili rischi per la sicurezza di chi guida il mezzo.
- **Manutenzione preventiva**: si basa su interventi programmati a intervalli regolari, senza considerare realmente le condizioni del veicolo. Questo approccio riduce il rischio di guasti improvvisi, ma potrebbe portare a interventi non necessari e anche ad una gestione inefficiente delle risorse.

Con l'arrivo dei **Big Data** e dell'apprendimento automatico (Machine Learning, **ML**) è stato possibile sviluppare un approccio più efficace, noto come **manutenzione predittiva**. Questo metodo sfrutta i dati storici e in tempo reale per prevedere guasti futuri e pianificare, quindi, in modo più preciso gli interventi di manutenzione. Le principali tecnologie alla base di quest'ultima sono:

- **Raccolta dati dai veicoli in movimento**: in questo periodo storico, tutti gli ultimi veicoli sono dotati di sensori IoT, centraline elettroniche e sistemi di telemetria che collezionano una grande quantità di dati in tempo reale.

- **Analisi e pre-elaborazione dei dati**: prima di essere utilizzati per il training dei modelli di ML, i dati devono essere processati, in particolare vanno puliti, normalizzati e trasformati in un formato utile per l'analisi seguente.
- **Modelli predittivi basati su ML e reti neurali**: grazie ad algoritmi avanzati si possono identificare pattern nei dati e prevedere con alta precisione quando un certo componente potrebbe guastarsi.
- **Integrazione con sistemi software scalabili:** per garantire efficienza e affidabilità, i modelli generati devono essere implementati in un architettura distribuita, in grado di gestire grandi richieste in tempo reale.

1.1.1 Benefici e impatti della manutenzione predittiva

L'adozione della manutenzione predittiva sta diventando sempre più cruciale per l'evoluzione dell'industria automobilistica. Proprio per questo, sempre più aziende stanno investendo in queste soluzioni basate su **Cloud Computing**, **Machine Learning** e **IoT**, rendendo necessario lo sviluppo di architetture software robuste in grado di integrare e gestire questi sistemi in modo efficiente. Tra i principali benefici dell'implementazione della manutenzione predittiva ci sono:

- **Riduzione dei costi operativi**: prevedendo in anticipo i guasti, le aziende possono pianificare interventi di manutenzione in modo ottimale, evitando costosi fermi macchina.
- **Miglioramento dell'efficienza operativa**: grazie alla gestione ottimizzata delle manutenzioni, le flotte di veicoli possono operare in modo continuo e affidabile, riducendo il numero di interruzioni non previste.
- **Aumento della sicurezza**: prevenire guasti critici aiuta a ridurre il numero di incidenti stradali ed aumenta la sicurezza dei conducenti.
- **Sostenibilità ambientale**: evitando interventi di manutenzione superflui e ottimizzando l'uso delle risorse, si può avere una gestione più sostenibile dei veicoli, riducendo sprechi e diminuire l'impatto ambientale.

1.1.2 Sfide e prospettive future

Nonostante i molti vantaggi introdotti dalla manutenzione predittiva, ha anche dei limiti legata alla qualità e alla disponibilità dei dati, in quanto il modello predittivo dipende fortemente dalla precisione e dalla completezza delle informazioni raccolte dai veicoli. Inoltre, l'integrazione di questi sistemi per l'azienda richiede investimenti significativi e una gestione efficace delle risorse IT. In ottica futura, l'evoluzione

delle tecnologie emergenti, come il **5G**, l'**edge computing** e l'**intelligenza artificiale avanzata**, permetterà di ottenere sistemi sempre più precisi, autonomi e intelligenti. In questa tesi la manutenzione predittiva è strettamente legata al settore dell'automotive, ma può essere estesa a molti altri settori come il trasporto pubblico, l'aeronautica, settore agricolo, settore manufatturiero ed altri.

1.1.3 L'azienda

L'azienda in cui ho sviluppato questo progetto è Veicoli S.R.L., parte del gruppo Endurance Engineering, operante nel settore automotive. L'azienda si occupa sia dello sviluppo di un prodotto interno sia di consulenze IT per clienti del settore. Nell'ambito della mia tesi, ho lavorato sui dati provenienti da Veicoli Business, un'applicazione sviluppata internamente e offerta a clienti come per esempio Univex, un'azienda specializzata nella logistica e nel trasporto di materiali sensibili, tra cui medicinali e organi. Il servizio principale che offriamo ai clienti è la gestione Fleet Management, basata su una centralina GeoTab installata su ogni veicolo della flotta. In quanto rivenditori ufficiali, non solo forniamo l'hardware, ma anche un'infrastruttura software avanzata che sfrutta i dati della centralina per ottimizzare la gestione dei veicoli. Il sistema di Fleet Management permette di monitorare in tempo reale lo stato dei veicoli, pianificare e analizzare le attività di manutenzione, gestire la flotta in modo efficiente e controllarne la posizione tramite GPS. La piattaforma offre reportistica avanzata e strumenti di analisi per migliorare l'efficienza operativa e ridurre i costi di gestione. Un ulteriore valore aggiunto è il servizio di monitoraggio delle temperature, fondamentale per aziende che necessitano di consegne a temperatura controllata. Grazie ai dati in tempo reale, il sistema garantisce il rispetto delle condizioni di trasporto richieste, evitando sprechi e preservando l'integrità dei prodotti trasportati.

VEICOLI
ENDURANCE GROUP

Figura 1.1: Logo Veicoli S.R.L

1.2 Obiettivi della tesi

L'obiettivo principale della tesi è sviluppare un sistema software in grado di eseguire la **manutenzione predittiva** dei veicoli mediante tecniche di **Machine Learning**.

In particolare, il progetto prevede le seguenti fasi:

1. **Raccolta e analisi dei dati storici e in tempo reale**:
   - Utilizzo di dataset contenenti informazioni sui veicoli, lo storico dei guasti e dati telemetrici.
   - Pre-elaborazione dei dati su **Google Colab**, includendo tecniche per pulire i dati, feature engineering e normalizzazione.
2. **Sviluppo di un modello predittivo**:
   - Creazione di un modello basato su reti neurali e tecniche di fine-tuning per migliorarne le prestazioni.
   - Addestramento del modello.
   - Test e valutazione delle performance del modello con metriche adeguate.
3. **Implementazione dell'architettura software**:
   - Separazione dell'applicazione in tre componenti principali, ciascuna containerizzata tramite **Docker**:
     - **API modello**: servizio dedicato all'inferenza del modello.
     - **Back-End Nest.js**: gestione della logica applicativa e della comunicazione con la UI.
     - **Front-End Vue.js**: interfaccia utente per visualizzare i risultati delle analisi.
4. **Deploy e scalabilità del sistema**:
   - Utilizzo di **Google Cloud Run** per il deploy delle tre componenti containerizzate.
   - Ottimizzazione delle prestazioni mediante **monitoraggio** e **logging**.

Il risultato finale sarà, quindi, un sistema in grado di prevedere guasti meccanici con un'elevata accuratezza , riducendo costi operativi e migliorando l'affidabilità dei veicoli della flotta.

1.3 Struttura della tesi

La tesi è suddivisa nei seguenti capitoli:

- **Capitolo 2 - Manutenzione Predittiva**
  - Presentazione delle tecnologie di Machine Learning .

  - Analisi delle architetture più utilizzate nell'automotive per la gestione dei dati e delle previsioni dei guasti.

- **Capitolo 3 - Architettura del sistema**
  - Descrizione dell'architettura software con focus sulle tecnologie utilizzate (Nest.js, Vue.js, Docker, Google Cloud Run).
  - Panoramica della gestione dei dati e della comunicazione tra i diversi moduli.

- **Capitolo 4 - Descrizione dei dati a disposizione**
  - Dettagli sul dataset, fasi di pre-processing, feature engineering, normalizzazione e trasformazione dei dati.
  - Descrizione delle scelte fatte in termini di feature.

- **Capitolo 5 - Sviluppo del modello di Machine Learning**
  - Approccio adottato per il training, fine-tuning e validazione del modello su Google Colab.

- **Capitolo 6 - Deploy e scalabilità del sistema**
  - Pipeline di deploy su Google Cloud Run e orchestrazione dei vari container docker.
  - Strategie di ottimizzazione delle risorse e monitoring delle prestazioni.

- **Capitolo 7 - Valutazione e analisi dei risultati**
  - Analisi delle performance del modello predittivo.

- **Capitolo 8 - Conclusioni e sviluppi futuri**
  - Sintesi dei risultati ottenuti e vantaggi dell'approccio adottato.
  - Possibili miglioramenti e sviluppi futuri.

Capitolo 2

Manutenzione Predittiva e Big Data

Nell'ultimo periodo, l'industria automobilistica ha assistito ad una trasformazione molto significativa grazie proprio all'integrazione delle tecnologie digitali nei processi di gestione e di manutenzione dei veicoli. Tra tutte le innovazioni più degne di nota, la **manutenzione predittiva** si è imposta come un metodo fondamentale per ridurre i costi operativi, minimizzare i tempi di fermo e migliorare l'affidabilità delle flotte di veicoli. L'arrivo dei **Big Data**, dell'**Internet of Things (IoT)** e dell'**Intelligenza Artificiale (AI)** hanno reso possibile lo sviluppo di algoritmi avanzati in grado di analizzare enormi quantità di dati (anche in tempo reale), offrendo delle previsioni precise in diversi ambiti. Nel nostro caso specifico si parla di previsioni sullo stato di salute delle parti meccaniche dei veicoli. Questo rispetto a prima segna un nuovo approccio alla manutenzione per i veicoli, non basandosi più su interventi reattivi o su programmi di revisione periodica. In questo capitolo verranno analizzati in dettaglio i concetti fondamentali della manutenzione predittiva, confrontandoli con gli approcci tradizionali, evidenziando le sue applicazioni nel settore automotive e approfondendo quali sono le maggiori sfide che le aziende devono affrontare per poter implementare sistemi di manutenzione predittiva efficaci.

2.1 Machine Learning

Il **Machine Learning** è una sottobranca dell'Intelligenza Artificiale che consente ai sistemi informatici di apprendere dai dati e prendere delle decisioni autonomamente. Questa disciplina ha rivoluzionato diversi settori, tra cui quello dell'automotive, dove viene utilizzata principalmente per sviluppare sistemi avanzati di manutenzione predittiva.

2.1.1 Tipologie di Machine Learning

Il Machine Learning si suddivide principalmente in tre categorie:

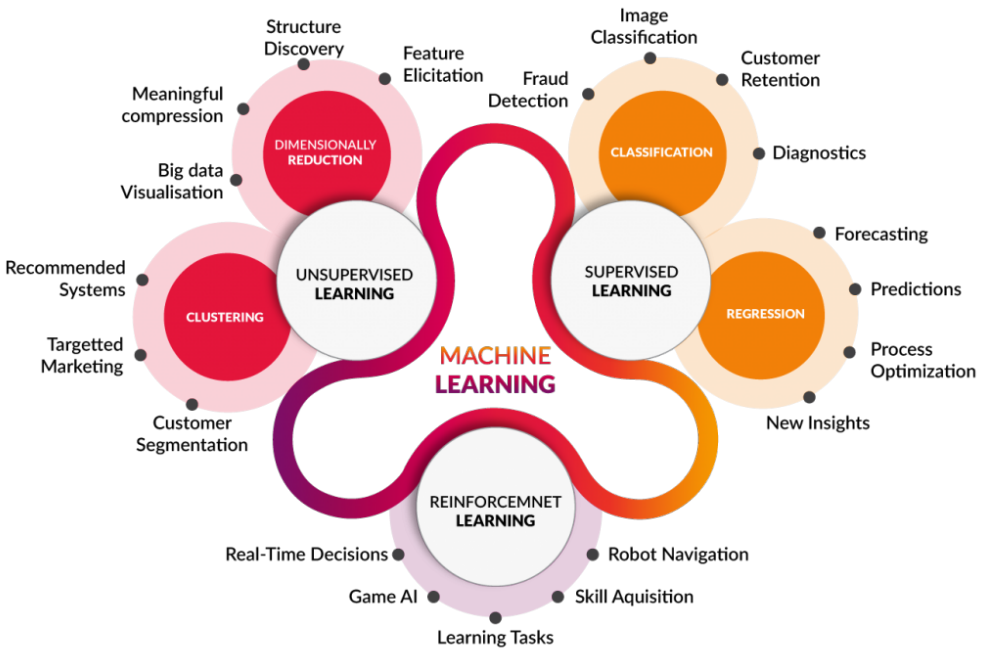

Figura 2.1: Tipologie di Machine Learning [1]

Apprendimento supervisionato (Supervised Learning)

L'apprendimento supervisionato utilizza dati etichettati, cioè dataset in cui ogni esempio è collegato con la risposta corretta, chiamata label. Il modello impara a generalizzare le relazioni tra input e output, per applicarle poi a nuovi dati. Alcuni esempi di questo approccio nell'ambito della manutenzione predittiva sono:

- **Regressione**: prevedere il tempo rimanente prima che un componente si guasti.
- **Classificazione**: identificare se un certo veicolo è a rischio di guasti analizzando i dati metrici forniti.

Apprendimento non supervisionato (Unsupervised Learning)

Nell'apprendimento non supervisionato, i dati non sono etichettati e il modello deve, quindi, identificare autonomamente pattern e strutture interne. Alcuni esempi di questo approccio nel campo dell'automotive sono:

- **Clustering**: raggruppare tutti i veicoli che hanno comportamenti simili per identificare degli schemi di usura.
- **Anomaly detection**: individuare componenti che mostrano segnali di usura diversi da quanto ci si aspetta.

Apprendimento per rinforzo (Reinforcement Learning)

L'apprendimento per rinforzo si basa su un sistema di ricompense e di penalità, il modello apprende ottimizzando ogni volta le sue azioni per massimizzare quello che è il risultato desiderato. Un esempio è:

- **Sistema di ottimizzazione carburante**: fornire indicazioni su come ottimizzare i consumi di carburante in base allo stile di guida.

2.2 Big Data

I Big Data sono un concetto abbastanza recente, in particolare nasce insieme all'era digitale e la conseguente esplosione della quantità di dati generati dai dispositivi elettronici e dagli esseri umani. Quest'ultimi sono dei grandi insiemi di dati che devono seguire un determinato e avanzato processo per poter essere elaborati e sfruttati al meglio [2]. Nel settore automotive, permettono di raccogliere e analizzare i dati provenienti dai veicoli per migliorare la qualità della predizione per la manutenzione.

2.2.1 Caratteristiche dei Big Data

I Big Data si distinguono per le **5V**:

Figura 2.2: 5V dei Big Data [3]

- **Volume**: grandi volumi di dati generati continuamente.
- **Velocità**: ricezione in tempo reale di flussi di dati.
- **Varietà**: dati provenienti da diverse fonti.

- **Veridicità**: qualità e affidabilità delle informazioni raccolte.
- **Valore**: capacità di estrarre informazioni utili.

2.3 IoT

L'Internet of Things è una rete di oggetti intelligenti dotati di sensori, connessi tramite Internet che consentono di comunicare tra di loro, spedendo e ricevendo dati [4]. Nel settore automotive, questo permette di ricevere in tempo reale i dati dai veicoli e successivamente elaborarli.

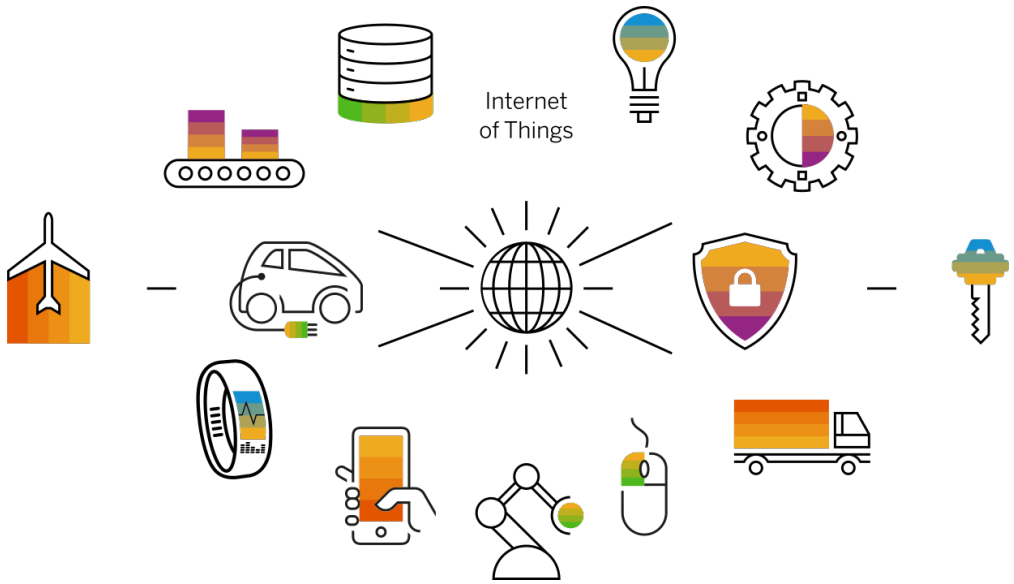

Figura 2.3: Schema IoT [4]

2.3.1 L'IoT nel settore automotive

Nel settore automobilistico, l'IoT ha portato alla creazione di veicoli intelligenti e connessi tra loro. Le principali fonti di dati sono:

- **Sensori di temperatura e vibrazioni**: monitorano l'usura delle parti meccaniche.
- **Centraline elettroniche (ECU)**: registrano anomalie ed errori.
- **Sistemi di telemetria**: trasmettono i dati sulle condizioni in tempo reale del veicolo e sullo stile di guida del conducente.

2.4 Definizione e importanza della manutenzione predittiva

La **manutenzione predittiva** permette di monitorare lo stato di salute di un componente del sistema, anticipando potenziali guasti grazie all'analisi di dati storici e in tempo reale. A differenza degli approcci precedenti, che prevedono interventi solo dopo il verificarsi di un guasto o secondo dei programmi prestabiliti, la manutenzione predittiva mira ad ottimizzare i tempi e i costi della manutenzione, intervenendo solo quando necessario. Quest'ultima è resa possibile grazie a diverse innovazioni tecnologiche, quali:

- **Internet of Things (IoT)**: i veicoli moderni sono dotati di **sensori avanzati**, centraline e dispositivi telemetrici che raccolgono dati sulle prestazioni dei vari componenti in tempo reale.
- **Big Data e Cloud Computing**: la capacità di gestire una grande quantità di dati, consente di raccogliere informazioni da diverse fonti, analizzarle e ricavarne dei **pattern**.
- **Machine Learning e Intelligenza Artificiale**: i modelli predittivi analizzano i dati per identificare segnali di usura, anomalie e prevedere la durata residua dei componenti.
- **Sistemi software scalabili**: sono fondamentali per elaborare e gestire queste informazioni, è necessario un ecosistema adeguato per gestire la mole di dati e fornire indicazioni ai gestori delle flotte.

L'importanza di quest'ultima nel settore dell'automotive è quindi evidente:

- **Riduzione dei costi operativi**: prevenire i guasti evita spese impreviste e interventi di emergenza.
- **Minimizzazione dei tempi di fermo**: il monitoraggio continuo permette di pianificare al meglio la manutenzione, senza compromettere la produttività del veicolo.
- **Miglioramento della sicurezza**: individua le anomalie riducendo il rischio di incidenti legati a guasti meccanici.
- **Ottimizzazione dell'utilizzo delle risorse**: le aziende possono gestire meglio i ricambi.

Grazie a questi vantaggi, la manutenzione predittiva, sta sempre prendendo più piede nelle aziende e c'è un crescente interesse ed investimento verso queste nuove tecnologie per la gestione ottimale delle flotte.

2.5 Differenze tra manutenzione correttiva, preventiva e predittiva

La manutenzione dei veicoli ha sempre seguito due principali approcci: **correttivo** e **preventivo**. La manutenzione predittiva rappresenta, quindi, un'evoluzione di questi modelli, portando con sè diversi vantaggi.

2.5.1 Manutenzione Correttiva

La manutenzione correttiva, viene detta anche "a guasto", consiste nell'intervenire sul veicolo solo quando un componente si rompe o smette di funzionare correttamente. È il metodo più semplice ed anche quello più immediato, ma ha di contro che è anche quello più costoso e rischioso.

Vantaggi:

- Nessun costo di manutenzione prima del guasto.
- Facile da implementare, non richiede un monitoraggio.

Svantagggi:

- Tempi di fermo imprevisti, con impatto negativo sulla produttività.
- Costi di riparazioni alti, specialmente se il guasto coinvolge più componenti.
- Possibili problemi di sicurezza, se il guasto avviene durante la guida per esempio.

2.5.2 Manutenzione Preventiva

La manutenzione preventiva si basa su un calendario prestabilito di interventi, senza tener conto dello stato effettivo dei componenti. Ad esempio, un veicolo potrebbe essere sottoposto a revisione ogni 20.000km, anche se tutti i suoi componenti sono ancora in buone condizioni.

Vantaggi:

- Riduce il rischio di guasti improvvisi.
- Migliora la pianificazione degli interventi.

Svantaggi:

- Può portare a sostituzioni inutili di componenti ancora funzionanti.
- Costi di manutenzione superiori rispetto alla manutenzione predittiva.

2.5.3 Manutenzione Predittiva

La manutenzione predittiva sfrutta il monitoraggio in tempo reale e l'analisi dei dati storici per ottimizzare gli interventi di manutenzione. Questo approccio permette di prevedere con elevata accuratezza quando un componente si sta avvicinando alla fine della sua vita, evitando guasti improvvisi senza dover effettuare sostituzioni inutili.

Vantaggi:

- Interventi mirati, basati sul reale stato dei componenti.
- Riduzione dei costi operativi, grazie ad una manutenzione più efficiente.
- Miglioramento della sicurezza, grazie alla prevenzione dei guasti.

Svantaggi:

- Richiede l'integrazione di tecnologie avanzate, come IoT nei veicoli, AI e Big Data nell'infrastruttura software.
- Può avere costi iniziali elevati, legati all'implementazione di sensori e infrastutture software.

2.6 Applicazioni della manutenzione predittiva nel settore automotive

L'industria dell'automotive ha visto una crescita esponenziale nell'utilizzo della manutenzione predittiva, soprattutto grazie ai progressi tecnologici degli ultimi tempi sia nel campo delle connessioni sia in ambito di elaborazione e studio di dati. Alcune delle applicazioni più rilevanti in questo ambito sono:

2.6.1 Monitoraggio della salute del motore

Tramite dei sensori installati all'interno dei motori, è possibile monitorare alcuni parametri critici come la temperatura, la pressione dell'olio, il numero di giri del motore, le vibrazioni, ecc. Successivamente i dati raccolti vengono poi analizzati da algoritmi avanzati di Machine Learning per individuare possibili segni di usura o delle anomalie nel funzionamento del motore.

2.6.2 Diagnostica avanzata per freni e sospensioni

Alcuni sistemi di manutenzione predittiva consentono di monitorare lo stato di usura dei freni e delle sospensioni analizzando i dati provenienti da accelerometri,

giroscopi e altri sensori. In questo modo, risulta possibile prevedere quando sarà necessario sostituire pastiglie o dischi dei freni e le sospensioni.

2.6.3 Gestione delle flotte aziendali

Le aziende che hanno una sezione di logistica o di trasporto utilizzano sistemi di manutenzione predittiva per ottimizzare la gestione delle proprie flotte. Questi sistemi permetto all'azienda di:

- **Pianificare interventi di manutenzione**, evitando fermi imprevisti.
- **Ridurre i costi operativi**, grazie alla manutenzione mirata.
- **Aumentare la durata e la sicurezza dei veicoli**, ottimizzando l'uso dei componenti e continuando a monitorarli.

2.6.4 Veicoli elettrici e batterie

Nel caso in cui un'azienda disponga di veicoli elettrici, la manutenzione predittiva è fondamentale per monitorare lo stato di salute del **pacco batterie** del veicolo, il quale è una delle componenti più delicato e costoso. Per far ciò si analizzano i dati provenienti dai sistemi di gestione delle batterie (**Battery Management Sistem - BMS**), in questo modo è possibile prevedere degradazione e malfunzionamenti, ottimizzando la sostituzione o la ricarica delle celle.

2.7 Sfide nell'implementazione di sistemi di manutenzione predittiva

Nonostante i benefici della manutenzione predittiva siano evidenti, esistono diverse sfide che un'azienda deve affrontare per implementare un sistema efficace:

- **Gestione di grandi quantità di dati**: i veicoli moderni generano un grande volume di dati, che devono essere poi elaborati in tempo reale per fornire delle previsioni accurate.
- **Costo iniziale elevato**: l'implementazione richiede un importante investimento in sensori, infrastruttura cloud e lo sviluppo del software.
- **Affidabilità dei modelli di Machine Learning**: i modelli dovrebbero essere costantemente aggiornati per adattarsi a nuove condizioni operative.

Tuttavia, la manutenzione predittiva rappresenta il futuro della gestione dei veicoli, offrendo grandi vantaggi concreti per le aziende.

Capitolo 3

Tecnologie Utilizzate

Lo sviluppo di questo sistema ha richiesto l'integrazione di diverse tecnologie, ciascuna aveva un ruolo specifico nel processo di acquisizione, elaborazione e visualizzazione dei dati. All'interno di questo capitolo verranno descritte in dettaglio le tecnologie utilizzate, le funzionalità e l'architettura.

3.1 Apache Spark: architettura e funzionalità

Apache Spark è un framework open-source per l'elaborazione distribuita di grandi volumi di dati (**Big Data**) in tempo reale. Quest'ultimo ha la capacità di elaborare dati in memoria, viene spesso impiegato in combinazione con i Big Data e con algoritmi di Machine Learning.

3.1.1 Architettura di Apache Spark

L'architettura di Apache Spark si basa su tre componenti fondamentali:

- **Driver Program**: controlla l'esecuzione dell'applicazione e distribuisce il carico di lavoro tra i nodi.
- **Cluster Manager**: gestisce le risorse e assegna i task ai vari worker. Può essere **Kubernetes**, **YARN** o **Standalone**.
- **Executor**: esegue le operazioni assegnate dal Driver e immagazzina i dati in cache per velocizzare il calcolo.

Spark utilizza un modello basato su **RDD (Resilient Distributed Dataset)**, che consente un'elaborazione fault-tolerant e distribuita.

3.1.2 Funzionalità principali di Apache Spark

Apache Spark si suddivide in 5 funzionalità principali:

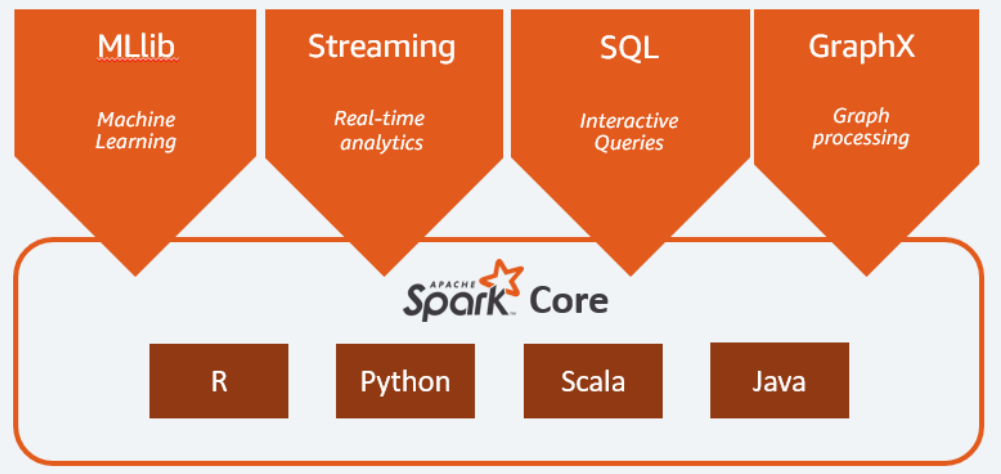

Figura 3.1: Le 5 funzionalità di Apache Spark[5]

1. **MLlib**: libreria per il Machine Learning distribuito.
2. **Spark Streaming**: permette l'analisi dati in real-time.
3. **Spark SQL**: modulo per l'elaborazione di dati strutturati con query SQL-like.
4. **GraphX**: utile per l'analisi di dati a grafo.
5. **Spark Core**: il "motore" centrale della piattaforma che gestisce la programmazione distribuita.

In questo progetto, Apache Spark è stato utilizzato per l'elaborazione e l'analisi dei dati telemetrici provenienti dai veicoli, con l'utilizzo, in particolare, di MLlib per la costruzione del modello predittivo.

3.2 MLlib: Machine Learning con Spark

MLlib è la libreria di Machine Learning di Apache Spark che permette di eseguire algoritmi di apprendimento automatico su dati distribuiti. Quest'ultima offre strumenti per:

- **Regressione e classificazione**, usate per prevedere il degrado delle parti dei veicoli.

- **Clustering**, usate per raggruppare veicoli con comportamenti simili.
- **Feature Engineering**, consiste nella selezione e trasformazione delle caratteristiche.

In questo progetto MLlib è stata fondamentale perchè è in grado di eseguire il **training in parallelo**, risultando efficiente pur lavorando con un grande dataset. Quest'ultima è stata utilizzata per costruire un modello di regressione, addestrato sui dati storici delle manutenzioni per poi prevedere il chilometraggio di vita rimanente prima della prossima manutenzione.

3.3 Google Colab: sviluppo e creazione del modello

Per la prototipazione del modello predittivo ho usato Google Colab, tecnologia che mi ha permesso di fare uno sviluppo cloud-based con GPU, in modo da velocizzare quanto più possibile la fase di fine-tuning e di training.

3.3.1 Acquisizione dei dati

I dati sono stati acquisiti tramite una centralina GeoTab, precisamente la GeoTab GO9+, la quale raccoglie direttamente le informazioni direttamente dal **CAN (Controller Area Network) Bus** dei veicoli. Quest'ultimo è il protocollo standard utilizzato dai veicoli per la comunicazione tra i diversi componenti elettronici, come motore, trasmissione, freni e sensori. La centralina estrae e trasmette in tempo reale:

- **ID Veicolo**
- **Chilometri totali percorsi**
- **Giri motore (RPM)**
- **Consumo di carburante**
- **Temperatura del motore**
- **Dati sui freni e accelerazione**
- **Errori diagnostici (DTC - Diagnostic Trouble Codes)**
- **Eventi di manutenzione programmata o straordinaria**

I dati vengono inviati a un **database cloud**, da cui vengono prelevati ed elaborati per la creazione del modello predittivo. Con i dati provenienti dal **CAN Bus** permette di avere un'analisi precisa dello stato del veicolo.

Figura 3.2: Centralina GeoTab GO9+[6]

3.3.2 Pre-processing dei dati

Prima di addestrare il modello, è stato necessario **pulire**, **normalizzare** e **integrare** i dati provenienti dalle due principali fonti di dati di cui disponiamo:

1. Il database con i dati storici delle manutenzioni e di tutte le informazioni dei veicoli.

2. I dati metrici acquisiti in tempo reale tramite le centraline GeoTab.

Preparazione dei dati storici

Per i dati storici, ho esportato le tabelle di interesse in formato **.csv**, caricate su Google Drive e successivamente lette in **Apache Spark** all'interno di un **DataFrame**. La prima operazione di pulizia ha riguardato i dati relativi alle parti meccaniche registrate nel database:

- Ho mantenuto esclusivamente le parti predefinite nell'applicazione, escludendo tutte le parti "custom" che avrebbero potuto introdurre rumore nel modello e compromettere la predizione.

- Ho creato un unico DataFrame, eseguendo dei join tra le tabelle per avere tutte le informazioni in un unico dataset strutturato.
- Ho selezionato le colonne ritenute più rilevanti come **feature** per il modello, eliminando le righe in cui almeno una di queste conteneva valori nulli.

Il risultato finale di questa prima fase è un DataFrame pulito e strutturato, contenente lo storico delle manutenzioni per ogni veicolo.

Acquisizione e preparazione dei dati metrici

I dati metrici, a differenza di quelli storici, sono stati acquisiti tramite **API GeoTab**, il servizio usato per raccogliere e gestire le informazioni acquisite dalle centraline installate nei veicoli. Il processo di acquisizione ha visto due fasi:

1. **Autenticazione**: una prima chiamata API che serve per **autenticarsi** nel database del cliente di interesse ed ottenere l'**access token**.
2. **Raccolta dei dati**: successivamente, viene eseguita una chiamata **API MultiCall** per estrarre tutti i dati telemetrici registrati nell'**anno 2024**, considerando tutti i veicoli appartenenti a quello specifico database.

Dopo l'acquisizione, i dati metrici sono stati aggregati per giorno, calcolando per ogni veicolo e per ogni metrica:

- **Valore medio giornaliero**
- **Valore massimo giornaliero**

Questa fase è stata fondamentale per ridurre di molto le dimensioni del dataset, senza questa aggregazione, avremmo avuto una registrazione ogni 10 secondi circa, rendendo il dataset eccessivamente grande e difficile da gestire ed utilizzare. Infine, i dati sono poi stati convertiti nelle unità di misura appropriate e salvati in formato **.csv** su Google Drive, con un **DataFrame separato** per ogni cliente.

Unione dei dati storici e metrici

Per addestrare il modello, è stato necessario combinare i **dati storici** con i **dati metrici** estratti dalle centraline. Questo è stato fatto eseguendo un join tra i due DataFrame, verificando la corrispondenza degli **ID Veicolo** e selezionando, per ogni manutenzione, le metriche dei **21 giorni precedenti**. Per gestire eventuali **valori nulli** o **mancanti** nei dati metrici, è stata adottata una strategia secondo la quale se una metrica presentava valore uguale a 0, questa veniva sostituita con la media dei valori precedenti per evitare che la mancanza di dati alterasse la predizione del modello. Dopo queste operazioni, il **DataFrame finale** è stato esportato e salvato in formato .csv per l'addrestramento del modello.

Feature Scaling e normalizzazione

L'ultimo passaggio prima dell'addestramento è stato il **Feature Scaling**, necessario per uniformare tutte le feature e renderle compatibili con **MLlib**. Il DataFrame finale è stato caricato in una variabile Spark, normalizzato e trasformato in un **feature vector**, pronto per essere utilizzato nel processo di training del modello predittivo.

3.3.3 Fine-Tuning e ottimizzazione del modello

Il modello è stato ottimizzando usando **Hyperparameter Tuning** con **Grid Search** per selezionare:

- **Numero di alberi** (nel caso di Random Forest).
- **Profondità massima degli alberi**.
- **Tasso di apprendimento** (nel caso di modelli di Gradient Boosting).

Grid Search è un metodo tradizionale usato nel Machine Learning, che consiste nel testare accuratamente tutte le combinazioni degli hyperparameter forniti al fine di trovare il miglior modello [7].

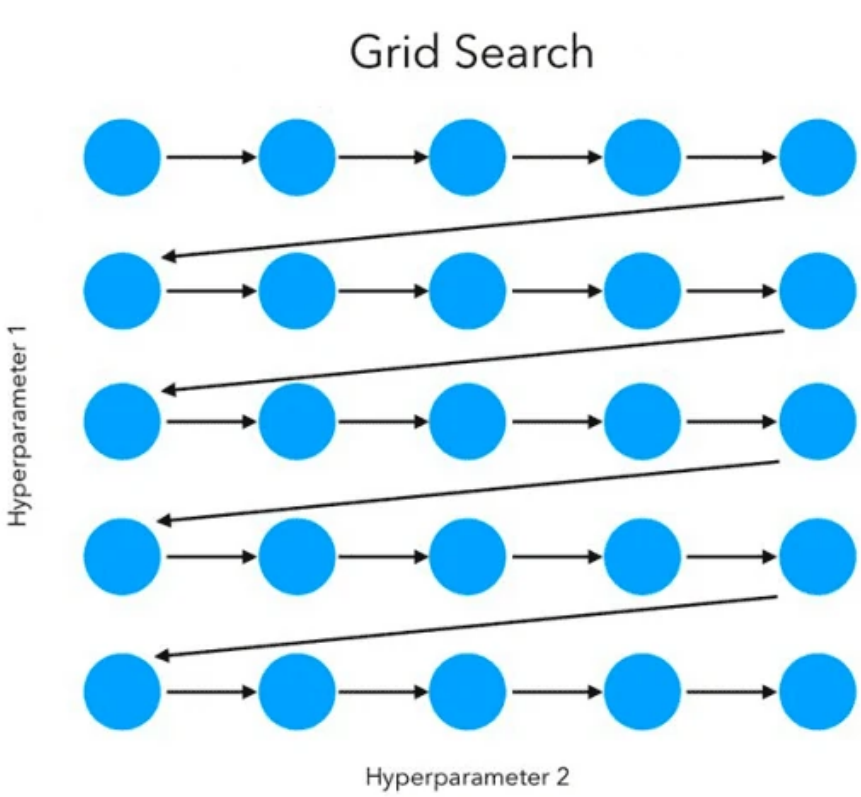

Figura 3.3: Funzionamento di Grid Search[8]

3.3.4 Training e valutazione delle performance

Il training è stato effettuato con **Spark MLlib**, dividendo il dataset in training e test set. Le metriche di valutazione sono state **RMSE (Root Mean Square Error)** e **MAE (Mean Absolute Error)**.

Modelli utilizzati e approccio ensemble

Per migliorare l'accuratezza delle previsioni, sono stati testati quattro modeli tradizionali:

1. **Random Forest**
2. **LightGBM (LGBM)**
3. **CatBoost**
4. **XGBoost**

Ognuno di questi modelli ha generato una previsione indipendente, che è stata poi utilizzata come feature di input per un **meta-modello** basato su XGBoost.

Stacking: utilizzo di un meta-modello XGBoost

Per migliorare ulteriormeente le performance del modello, ho adottato un approccio di **stacking**.

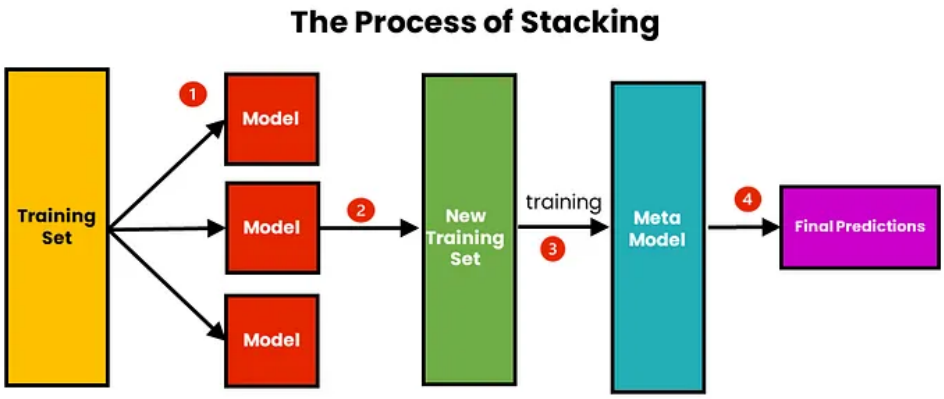

Figura 3.4: Schema di funzionamento dello Stacking[9]

Quest'ultimo è una tecnica di ensemble learning in cui le predizioni di più modelli vengono utilizzate come input per un **meta-modello**, che apprende a combinare i risultati in modo ottimale. In questo caso, il meta-modello scelto è **XGBoost**, che prende in input le previsioni dei tre modelli di base e le combina per produrre una previsione finale più accurata. Se l'RMSE del meta-modello risulta essere inferiore rispetto a quello dei singoli modelli, significa che l'approccio da noi scelto ha migliorato le prestazioni del sistema, riducendo l'errore e fornendo una stima più precisa del chilometraggio rimnente prima della prossima manutenzione. L'uso del meta-modello ha permesso di ottenere un modello più robusto, capace di combinare i punti di forza dei diversi algoritmi e di migliorare la qualità delle previsioni rispetto all'uso del singolo modello.

3.3.5 Esportazione del modello per l'integrazione nel sistema

Dopo il training e la valutazione delle prestazioni, i modelli sono stati salvati in formato **.pkl (Pickle)**. Questo formato consente di **serializzare** e **deserializzare** gli oggetti Python, facilitando il caricamento e l'integrazione dei modelli all'interno dell'**API Flask** containerizzata con **Docker**. Sono stati salvati:

- **I modelli di base**: Random Forest, LightGBM, CatBoost e XGBoost.
- **Il meta-modello**: XGBoost utilizzato nello stacking

3.3.6 Caricamento e utilizzo del modello nell'API Flask

All'interno dell'API, i modelli vengono caricati in memoria usando la libreria **loblib**, che permette un caricamento efficiente dei file .pkl. Durante le richieste API, i dati di input vengono preprocessati e passati ai modelli per ottenere una previsione del chilometraggio rimanente prima della prossima manutenzione. Questo approccio permette una **scalabilità** elevata e una facile **integrazione** con altri servizi, rendendo il sistema pronto per l'utilizzo su **Google Cloud Run**. In particolare sarà il server Nest.js a fare una richiesta POST all'API con al suo interno il modello, specificando nel corpo della richiesta i parametri ricevuti e calcolati:

```
{
    "geotab_vehicle_id": "b173",
    "geotab_name": "univex",
    "current_mileage": 270000,
    "AvgEngineOilTemperature": 80,
    "AvgCoolantTemperature": 100,
    "AvgOilPressure": 5.5,
    "AvgEngineSpeed": 2300,
    "MaxEngineOilTemperature": 93,
    "MaxCoolantTemperature": 120,
    "MaxOilPressure": 7,
    "MaxEngineSpeed": 4500
}
```

La risposta dell'API ha questo formato:

```
[
    {
        "part_id": 126,
        "part_name": "filtro_olio",
        "plate": "GD439MA",
        "prediction": 22394.7177734375,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 847,
        "part_name": "filtro_aria",
        "plate": "GD439MA",
        "prediction": 22394.7177734375,
        "version": "DUCATO (1994)"
```

```
    },
    {
        "part_id": 128,
        "part_name": "pastiglie_freni_anteriori",
        "plate": "GD439MA",
        "prediction": 22366.646484375,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 397,
        "part_name": "pastiglie_freni_posteriori",
        "plate": "GD439MA",
        "prediction": 22366.646484375,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 520,
        "part_name": "olio_motore",
        "plate": "GD439MA",
        "prediction": 5438.861328125,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 533,
        "part_name": "kit_distribuzione",
        "plate": "GD439MA",
        "prediction": 6199.8046875,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 396,
        "part_name": "filtro_abitacolo",
        "plate": "GD439MA",
        "prediction": 6199.8046875,
        "version": "DUCATO (1994)"
    },
    {
        "part_id": 401,
        "part_name": "dischi_freno_anteriori",
        "plate": "GD439MA",
        "prediction": 51285.4853515625,
        "version": "DUCATO (1994)"
    }
]
```


Il server la riceverà ed elaborerà le informazioni rendendole fruibili all'utente tramite UI.

3.4 Docker: conteinerizzazione e orchestrazione dei servizi

Docker è stato utilizzato containerizzare i diversi componenti del sistema, garantendo: **isolamento**, **scalabilità** e **portabilità** tra ambienti di sviluppo e produzione. Tuttavia, invece di orchestrare i container con Docker Compose in locale, i servizi sono stati pubblicati su **Google Cloud Run**, consentendo un deploy indipendente e un accesso pubblico ma **controllato**.

3.4.1 Strutturazione dei container

Il sistema è suddiviso in tre container principali, ognuno di questi poi costruito e caricato su **Google Artifact Registry**, per poi essere distribuito su Google Cloud Run:

- **API Flask**
  - Esegue le predizioni del modello di manutenzione predittiva.
  - Carica i modelli dai file .pkl e li rende risponibili tramite un endpoint REST.
  - Espone un'API accessibile solo dal server Nest.js per garantire sicurezza e controllo.
- **Server Nest.js**
  - Funziona come backend principale del sistema.
  - Gestisce l'interazione tra UI e API Flask, assicurando il rispetto delle policy di accesso.
  - Implementa validazioni, autenticazione e gestione delle richieste provenienti dalla UI.
- **UI Vue.js**:
  - Fornisce l'interfaccia utente per visualizzare i dati.
  - Effettua chiamate solo al server Nest.js, senza comunicare direttamente con l'API Flask.

3.4.2 Comunicazione tra container

Essendo distribuiti su Google Cloud Run, i servizi comunicano tra loro utilizzando i rispettivi **URL pubblici**, ma con restrizioni **CORS** ben definite per limitarne l'accesso:

- L'API Flask può essere contatatta solo dal server Nest.js.
- Il server Nest.js può essere contattato solo dalla UI Vue.js.
- La UI Vue.js non ha accesso diretto all'API Flask per evitare chiamate non autorizzate.

Questo approccio offre diversi vantaggi, come per esempio:

- **Maggior sicurezza**, poichè solo i servizi autorizzati possono comunicare tra loro.
- **Scalabilità**, ogni componente può essere ridimensionato indipendentemente.
- **Semplicità di deploy**, grazie a Google Cloud Run, che gestisce automaticamente il bilanciamento del carico e la disponibilità dei servizi.

3.5 Google Cloud Run: deploy e scalabilità in cloud

Per la distribuzione dell'intero sistema, è stato scelto **Google Cloud Run**, un servizio serverless che permette di eseguire container in un ambiente scalabile e gestito automaticamente.

3.5.1 Motivazioni della scelta

Google Cloud Run è stato scelto per i seguenti vantaggi:

- **Scalabilità automatica**, il sistema adatta il numero di istanze dei container in base alla richiesta, garantendo però sempre efficienza e riduzione dei costi.
- **Deploy semplificato**, il deploy avviene tramite un singolo comando da terminale.
- **Isolamento e sicurezza**, ogni servizio ha il proprio URL pubblico, gli accessi sono però regolati da CORS e restrizioni di rete.
- **Facilità di gestione**, non è necessario configurare nessun orchestratore, Google Cloud Run gestisce tutto in modo autonomo.

3.5.2 Configurazione e gestione del servizio

Il **deploy** dei servizi viene effettuato manualmente tramite **Google Cloud Run CLI**, seguendo i seguenti passaggi:

1. **Build dell'immagine Docker**
   - Per ogni modifica al codice, l'immagine del container viene aggiornata con il comando:

```
docker build -t gcr.io/<PROJECT_ID>/<SERVICE_NAME> .
```

2. **Push dell'immagine su Google Artifact Registry**
   - Dopo il build, l'immagine viene caricata sul registro di Google:

```
docker push gcr.io/<PROJECT_ID>/<SERVICE_NAME>
```

3. **Deploy del container su Google Cloud Run**
   - Il servizio viene aggiornato eseguendo:

```
gcloud run deploy <SERVICE_NAME> \
--image gcr.io/<PROJECT_ID>/<SERVICE_NAME> \
--region <REGION> \
--platform managed \
--allow-unauthenticated \
--memory= <MEMORY>
```

4. **Gestione degli accessi e restrizioni CORS**
   - **L'API Flask è accessibile solo al server Nest.js**, impedendo richieste da fonti esterne.
   - **Il server Nest.js accetta solo richieste dalla UI Vue.js**, proteggendo gli endpoint da accessi non autorizzati.

Questo approccio consente di controllare manualmente ogni deploy, lasciando l'architettura semplice.

3.6 Nest.js: back-end modulare per la gestione delle API

Per il back-end del sistema è stato utilizzato **Nest.js**, un framework TypeScript per Node.js che fornisce un'architettura modulare e scalabile.

3.6.1 Architettura del server

Il server Nest.js è strutturato in più moduli, ognuno con una responsabilità specifica:

- **Gestione utenti**, gestisce l'utenticazione, autorizzazione e gestione degli account.
- **Comunicazione con il modello predittivo**, interfacciamento con l'API Flask per ottenere le previsioni sui veicoli.
- **Loggin e sicurezza**, tracciamento delle richieste e restrizioni CORS per limitare l'accesso alle API.

3.6.2 Comunicazione con il modello predittivo

Nest.js funge da intermediario tra la UI e l'API Flask, seguendo questa logica:

1. La UI Vue.js invia una richiesta al server Nest.js tramite REST API.
2. Nest.js valida la richiesta e la inoltra all'API Flask, che esegue la predizione.
3. L'API Flask risponde con il risultato della predizione.
4. Nest.js elabora la risposta e la restituisce alla UI.

Per garantire la sicurezza, Nest.js imposta restrizioni CORS e accetta richieste solo dall'interfaccia front-end autorizzata. Inoltre, l'API Flask è accessibile solo dal server Nest.js, impedendo richieste dirette da fonti esterne.

3.7 Vue.js: interfaccia utente e visualizzazione dei dati

Vue.js è stato scelto per la UI con l'obiettivo di rendere l'app facilmente integrabile nei sistemi già esistenti.

3.7.1 Struttura della UI

L'interfaccia è suddivisa in quattro sezioni principali:

- **Login**, in cui l'utente si autentica.
- **Dashboard**, in cui si visualizza una tabella con le previsioni di manutenzione dei veicoli, più alcuni dettagli nella parte superiore di quest'ultima.
- **Sezione Dettagli Flotta**, in cui si visualizza la stessa tabella di prima in primo piano.
- **Sezione Dettaglio Veicolo**, in cui si visualizzano i dettagli del veicolo selezionato, una tabella con tutte le parti soggette a manutenzione e anche il grafico di quest'ultime.

3.7.2 Utilizzo di PrimeVue per la UI

Per la gestione dei componenti UI ho utilizzato **PrimeVue**, che offre componenti predefiniti, riducendo il tempo di sviluppo e migliorando la grafica. Ho utilizzato ques'ultimo anche perchè abbiamo pensato ad una futura integrazione nella nostra webapp principale, che è anch'essa sviluppata in Vue.js e con PrimeVue. In particolare, i componenti che ho utilizzato sono:

- **p-datatable**, per la visualizzazione dei dati in tabelle interattive.
- **p-chart**, per la rappresentazione grafica delle previsioni.
- **p-toast**, per le notifiche e per i feedback utente.

3.7.3 Interazione con il back-end

L'UI comunica con il server Nest.js tramite **Axios**, e per ridurre il numero di richieste ripetute e migliorare le performance generali, utilizzo **localStorage** come cache. In questo modo evito chiamate API ripetitive, visto che i dati vengono aggiornati giornalmente, non necessitando di un'aggiornamento continuo. Quando l'applicazione viene aperta dall'utente, viene verificato se le predizioni sono già presenti in **localStorage**, nel caso in cui questi siano presenti e sono aggiornate (meno di 24 ore) allora vengono usati i dati della cache. In caso contrario invece, la UI fa richiesta al back-end Nest.js, aggiorna i dati e li salva nel localStorage.

3.8 Pipeline e orchestrazione

L'orchestrazione dei servizi garantisce un flusso di dati fluido tra le varie componenti del progetto.

3.8.1 Flusso dei dati tra API modello, server e UI

L'interazione tra i vari componenti, segue il seguente schema:

1. **UI (Vue.js)** invia una richiesta al **server Nest.js**.
2. **Nest.js** inoltra una richiesta all'**API Flask**, che esegue la predizione.
3. **Flask** restituisce il risultato a **Nest.js**, che lo trasmette alla UI.
4. **Vue.js** aggiorna i grafici, utilizzando la cache locale per ottimizzare il numero di richieste.

3.8.2 Strategie di gestione degli aggiornamenti del modello

Per garantire aggiornamenti senza downtime, utilizzo:

- **Deploy separati su Google Cloud**
  - Ogni container è caricato su **Google Cloud Artifact Registry** e viene deployato singolarmente su Cloud Run. Questo permette di aggiornare un singolo componente senza andare ad interrompere gli altri.
- **Versionamento del modello con file .pkl**
  - Ogni modello viene salvato in formato .pkl per consentire un rapido rollback in caso di problemi. Il server carica automaticamente l'ultima versione disponibile senza necessità di downtime.
- **Gestione manuale degli aggiornamenti via terminale**
  - Aggiorno i container manualmente via **CLI**, mantenendo il controllo diretto su ogni modifica.

Capitolo 4

Dataset e Pre-elaborazione

In questo capitolo vengono descritti i dataset di cui ho fatto uso e delle operazioni che ho eseguito su quest'ultimi per ottenere un singolo dataframe con cui dare inizio alla fase di addestramento del modello.

4.1 Descrizione dei dataset

Il dataset utilizzato per l'addestramento del modello è stato costruito partendo da 2 fonti principali:

1. **Database Veicoli**, il quale contiene 5 tabelle principali:
    - **vehicles**, che contiene informazioni sui veicoli.
    - **vehicle_services**, che contiene le corrispondenze tra manutenzione e veicolo.
    - **vehicle_service_checks**, che contiene le corrispondenze tra manutenzione e parti cambiate.
    - **vehicle_service_parts**, che contiene informazioni sulle parti.
    - **vehicle_models**, che contiene le corrispondenze veicolo-modello.
    - **users**, che contiene le informazioni sul cliente.
2. **Database Geotab**, che raccoglie dati telemetrici in tempo reale tramite la centralina connessa al CAN bus.

Questi dati sono integrati e preprocessati per ottenere un unico dataset adatto poi per l'addestramento del modello di manutenzione predittiva.

4.1.1 Database veicoli

Il database Veicoli è composto da:

- Tabella **vehicles**, che contiene informazioni come:
  - ID del veicolo,
  - ID del modello,
  - ID utente,
  - tipo di motorizzazione,
  - targa,
  - fornitore gps,
  - ID gps
  - ecc..
- Tabella **vehicle_services**, che contiene informazioni come:
  - ID del servizio di manutenzione,
  - ID del veicolo,
  - km totali al servizio,
  - data servizio,
  - costo servizio,
  - ecc..
- Tabella **vehicle_service_checks**, che contiene informazioni come:
  - ID servizio di manutenzione,
  - ID parte sostituita,
  - quantità parte sostituitita
- Tabella **vehicle_service_parts**, che contiene informazioni come:
  - ID parte,
  - nome parte,
  - tipo parte
- Tabella **users**, che contiene informazioni come:
  - ID utente,
  - tipo di piano,

- nome geotab database,
- ecc..

L'unione di queste tabelle, consente di avere la storia completa di ogni veicolo, collegando il suo utilizzo alla frequenza delle manutenzioni.

4.1.2 Database di Geotab

Il database Geotab raccoglie i dati telemetrici in tempo reale dai veicoli tramite la centralina connessa al CAN bus. Le metriche raccolte includono:

- Temperatura olio motore,
- Temperatura liquido refrigerante motore,
- Pressione olio,
- Numero di giri motore,
- Numero di kilometri totali

I dati vengono acquisiti con una frequenza di circa una rilevazione ogni 10 secondi, rendendo necessaria quindi un'aggregazione giornaliera per ridurre la dimensione del dataset. Per poter prelevare i dati dal database GeoTab ho dovuto fare una chiamata API che richiedesse le metriche che si vogliono per tutti i veicoli di un determinato cliente. La chiamata API è una POST al seguente URL: *https://my.geotab.com/apiv1* ed è strutturata nel seguente modo:

```
{
    "method":"ExecuteMultiCall",
    "params":{
        "calls": [
            {
                "method": "Get",
                "params":{
                    "typeName": "StatusData",
                    "search":{
                        "diagnosticSearch":{"id":"
    DiagnosticEngineOilTemperatureID"},
                        "fromDate":"2025-03-19T00:00:00.000Z",
                        "toDate":"2025-03-20T00:00:00.000Z"
                    }
                }
            },
            {
```

```
                "method": "Get",
                "params":{
                    "typeName": "StatusData",
                    "search":{
                        "diagnosticSearch":{"id":"
DiagnosticEngineCoolantTemperatureID"},
                        "fromDate":"2025-03-19T00:00:00.000Z",
                        "toDate":"2025-03-20T00:00:00.000Z"
                    }
                }
            },
            {
                "method": "Get",
                "params":{
                    "typeName": "StatusData",
                    "search":{
                        "diagnosticSearch":{"id":"
DiagnosticOilPressureId"},
                        "fromDate":"2025-03-19T00:00:00.000Z",
                        "toDate":"2025-03-20T00:00:00.000Z"
                    }
                }
            },
            {
                "method": "Get",
                "params":{
                    "typeName": "StatusData",
                    "search":{
                        "diagnosticSearch":{"id":"
DiagnosticEngineSpeedId"},
                        "fromDate":"2025-03-19T00:00:00.000Z",
                        "toDate":"2025-03-20T00:00:00.000Z"
                    }
                }
            },
            {
                "method": "Get",
                "params":{
                    "typeName": "StatusData",
                    "search":{
                        "diagnosticSearch":{"id":"
DiagnosticRawOdometerId"},
                        "fromDate":"2025-03-19T00:00:00.000Z",
                        "toDate":"2025-03-20T00:00:00.000Z"
                    }
                }
            }
        ],
        "credentials": {
```

```
            "database": "univex",
            "sessionId": sessionID,
            "userName": username
        }
    }
}
```

Questa estrae le metriche specificate prima per ogni veicolo della flotta del cliente, che è specificato dal campo "database".

4.1.3 Unione e pulizia dei dati

Per costruire il dataset finale, ho dovuto integrare le informazioni provenienti da entrambi i database.

Unione dei dati

L'integrazione dei dati è avvenuta attraverso una serie di operazioni di join e trasformazioni per creare un dataset strutturato:

1. **Aggregazione dei dati storici**
   - Viene fatto il join tra *vehicles* e *vehicle_services* tramite ID del veicolo.
   - Viene fatto il join tra *vehicle_services* e *vehicle_service_checks* tramite ID del servizio di manutenzione.
   - Viene fatto il join tra *vehicle_service_parts* e *vehicle_service_checks* tramite ID parte.
   - Viene fatto il join tra *vehicle_models* e *vehicles* tramite ID del modello.
   - Viene fatto il join tra *vehicles* e *users* tramite ID utente.
   - Vengono poi presi solo gli utenti con tipo di piano "business" e i veicoli con fornitore gps "geotab".

   Dopo queste operazioni ottengo un dataframe di questo tipo:

2. **Aggregazione dei dati telemetrici**
   - I dati di Geotab, raccolti a intervalli di tempo molto ravvicinati, sono **aggregati giornalmente** per ridurre il volume dei dati e mantenere solo gli indicatori più rilevanti.
   - Per ogni veicolo e per ogni giorno, sono stati calcolati **valori medi** e **massimi** di alcune variabili come:
     – Temperatura olio motore,

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	vehicleid	plate	fuel	modelid	userid	gps_supplier	gps_serial_number	serviceid	km	date	partid	quantity	name	geotab_database	plan_type
2	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	128	1	filtro_aria	univex	business
3	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	126	1	filtro_olio	univex	business
4	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	533	1	filtro_abitacolo	univex	business
5	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	401	1	olio_motore	univex	business
6	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	396	1	pastiglie_freni_anteriori	univex	business
7	142074	FR706MJ	gas	739	193573	geotab	b71	102273	51037	2019-03-19	920	1	pastiglie_freno_posteriori	univex	business
8	130369	FC320MM	gas	76203	127540	geotab	b15	102275	266609	2019-03-18	807	1	bielletta_stabilizzatrice	univex	business
9	142071	FR703MJ	gas	739	193573	geotab	b486	95498	50886	2018-11-29	401	1	olio_motore	univex	business
10	142071	FR703MJ	gas	739	193573	geotab	b486	95498	50886	2018-11-29	126	1	filtro_olio	univex	business
11	142071	FR703MJ	gas	739	193573	geotab	b486	95498	50886	2018-11-29	129	1	filtro_aria_condizionata	univex	business
12	134835	FM354SL	gpl	75819	193573	geotab	b5	99868	116105	2019-02-07	401	1	olio_motore	univex	business
13	134835	FM354SL	gpl	75819	193573	geotab	b5	99868	116105	2019-02-07	126	1	filtro_olio	univex	business
14	134835	FM354SL	gpl	75819	193573	geotab	b5	99868	116105	2019-02-07	128	1	filtro_aria	univex	business
15	134835	FM354SL	gpl	75819	193573	geotab	b5	99868	116105	2019-02-07	129	1	filtro_aria_condizionata	univex	business
16	134835	FM354SL	gpl	75819	193573	geotab	b5	99868	116105	2019-02-07	396	1	pastiglie_freni_anteriori	univex	business
17	137790	EV849VL	diesel	76203	193573	geotab	b72	95500	335421	2018-11-28	401	1	olio_motore	univex	business
18	137790	EV849VL	diesel	76203	193573	geotab	b72	95500	335421	2018-11-28	126	1	filtro_olio	univex	business
19	137790	EV849VL	diesel	76203	193573	geotab	b72	95500	335421	2018-11-28	128	1	filtro_aria	univex	business
20	137790	EV849VL	diesel	76203	193573	geotab	b72	95500	335421	2018-11-28	129	1	filtro_aria_condizionata	univex	business
21	143151	FN733VF	diesel	36707	193573	geotab	b79	103175	92000	2019-04-01	126	1	filtro_olio	univex	business
22	143151	FN733VF	diesel	36707	193573	geotab	b79	103175	92000	2019-04-01	401	1	olio_motore	univex	business
23	130370	FC321MM	gas	76203	127540	geotab	b3B	95632	219400	2018-11-28	847	1	dischi_freno_anteriori	univex	business
24	130376	FC240YW	diesel	474	127540	geotab	b47	95736	243426	2018-11-30	401	1	olio_motore	univex	business
25	130376	FC240YW	diesel	474	127540	geotab	b47	95736	243426	2018-11-30	396	1	pastiglie_freni_anteriori	univex	business
26	137964	FG465EE	diesel	15606	193573	geotab	b8E	102272	201180	2019-03-19	397	1	pastiglie_freni_posteriori	univex	business
27	136637	EN210DY	diesel	76101	193573	geotab	b4D	97280	442112	2018-12-29	397	1	pastiglie_freni_posteriori	univex	business
28	137583	FN372VB	gas	739	193573	geotab	b6B	97295	89578	2018-12-28	401	1	olio_motore	univex	business
29	137583	FN372VB	gas	739	193573	geotab	b6B	97295	89578	2018-12-28	126	1	filtro_olio	univex	business
30	137583	FN372VB	gas	739	193573	geotab	b6B	97295	89578	2018-12-28	533	1	filtro_abitacolo	univex	business
31	137583	FN372VB	gas	739	193573	geotab	b6B	97295	89578	2018-12-28	397	1	pastiglie_freni_posteriori	univex	business
32	130408	FK936MA	diesel	474	193573	geotab	b6	97297	156875	2018-12-31	126	1	filtro_olio	univex	business
33	130408	FK936MA	diesel	474	193573	geotab	b6	97297	156875	2018-12-31	128	1	filtro_aria	univex	business
34	142073	FR704MJ	gas	739	193573	geotab	b78	97346	46698	2019-01-02	128	1	filtro_aria	univex	business
35	142073	FR704MJ	gas	739	193573	geotab	b78	97346	46698	2019-01-02	126	1	filtro_olio	univex	business

Figura 4.1: History DataFrame

- Temperatura liquido refrigerante motore,
- Pressione olio,
- Numero giri motore,
- Kilometri totali (solo il massimo, ossia il numero attuale)

Dopo queste operazioni ottengo un dataframe di questo tipo per ogni cliente:

	A	B	C	D	E	F	G	H	I	J	K
1	deviceID	date	AvgEngineOilTemperature	MaxEngineOilTemperature	AvgCoolantTemperature	MaxCoolantTemperature	AvgOilPressure	MaxOilPressure	AvgEngineSpeed	MaxEngineSpeed	current_mileage
2	b102	2024-01-02	76.79710145	98	78.93506494	100	3.164181022	5.432093023	1669.471115	3675.5	273930
3	b102	2024-01-03	80.66666667	101	81.73255814	103	3.164181022	5.432093023	1624.962338	3133	274006
4	b102	2024-01-04	82.97222222	101	83.23655914	99	3.164181022	5.432093023	1624.328879	3647.5	274096
5	b102	2024-01-05	81.48101266	99	81.71428571	99	3.164181022	5.432093023	1598.088357	3272	274161
6	b102	2024-01-08	85.70707071	99	84.80833333	102	3.164181022	5.432093023	1595.191957	3906.5	274267
7	b102	2024-01-09	79.39473684	99	80.71084337	101	3.164181022	5.432093023	1609.997784	4334	274330
8	b102	2024-01-10	79.53333333	99	80.89583333	100	3.164181022	5.432093023	1638.986223	3349	274421
9	b102	2024-01-11	77.74324324	99	78.90361446	99	3.164181022	5.432093023	1599.296838	3230.5	274488
10	b102	2024-01-12	79.20289855	98	81.01219512	99	3.164181022	5.432093023	1577.892308	3195.5	274551
11	b102	2024-01-13	74.890625	94	75.14705882	96	3.164181022	5.432093023	1683.314096	3902.5	274621
12	b102	2024-01-14	34	63	30.66666667	61	3.164181022	5.432093023	1756.514286	2946.5	274622
13	b102	2024-01-15	81.73611111	99	81.12195122	99	3.164181022	5.432093023	1635.658672	3520	274702
14	b102	2024-01-16	80.18309859	98	82.24390244	101	3.164181022	5.432093023	1576.782022	3817.5	274768
15	b102	2024-01-17	83.33846154	101	84.8	103	3.164181022	5.432093023	1569.636136	3067	274878
16	b102	2024-01-18	82.18309859	100	81.98795181	102	3.164181022	5.432093023	1589.657049	3527	274942
17	b102	2024-01-19	84.74324324	100	83.48809524	102	3.164181022	5.432093023	1591.488576	3583	275009
18	b102	2024-01-20	71.55769231	100	75.05660377	99	3.164181022	5.432093023	1649.316919	3328.5	275066
19	b102	2024-01-22	79.09375	98	80.30666667	99	3.164181022	5.432093023	1600.321127	3252	275154
20	b102	2024-01-23	76.73015873	96	78.25	99	3.164181022	5.432093023	1593.398133	3482	275230
21	b102	2024-01-24	79.171875	96	80.47619048	99	3.164181022	5.432093023	1562.660059	3978	275330
22	b102	2024-01-25	75.41509434	96	77.29230769	100	3.164181022	5.432093023	1548.289519	3675.5	275432
23	b102	2024-01-26	74.6875	99	78.61038961	101	3.164181022	5.432093023	1542.649588	3403	275506
24	b102	2024-01-27	73.48979592	98	71.94444444	93	3.164181022	5.432093023	1804.441848	3734.5	275680
25	b102	2024-01-28	53.3	91	51.4	87	3.164181022	5.432093023	1509.445455	2924	275686
26	b102	2024-01-29	78.75	97	81.25301205	100	3.164181022	5.432093023	1551.700059	3296	275771
27	b102	2024-01-30	78	99	78.94047619	101	3.164181022	5.432093023	1568.707666	3163	275857
28	b102	2024-01-31	76.54054054	98	79.28409091	100	3.164181022	5.432093023	1636.109233	3540	275943
29	b103	2024-01-02	96.65151515	114	78.36434109	91	3.164181022	5.432093023	1660.476644	3659	330694
30	b103	2024-01-03	93.96923077	112	78.80451128	92	3.164181022	5.432093023	1660.806873	3692	331044
31	b103	2024-01-04	95	114	78.56060606	92	3.164181022	5.432093023	1683.842154	3473	331389
32	b103	2024-01-05	92.94	113	78.58015267	91	3.164181022	5.432093023	1677.367929	3973	331753
33	b103	2024-01-06	91.16666667	112	75.73214286	92	3.164181022	5.432093023	1624.570646	3268	331942
34	b103	2024-01-07	76.57142857	102	63.66666667	77	3.164181022	5.432093023	1462.004237	3357	331953
35	b103	2024-01-08	82	116	70.875	94	3.164181022	5.432093023	1558.579655	3223	332072
36	b119	2024-01-03	74.55555556	91	72.48275862	89	3.164181022	5.432093023	1649.140844	3645.810916	160050

Figura 4.2: GeoTab DataFrame

3. **Creazione tabella finale**

- Il dataset è strutturato in modo che ogni riga rappresenti lo stato del veicolo nei 21 giorni precedenti ad un evento di manutenzione.

- Per ogni record di manutenzione sono stati selezionati i dati telemetrici delle tre settimane precedenti, andando così a creare una finestra temporale che permette di osservare l'andamento dei parametri prima del guasto.

4.1.4 Trattamento dei valori null e outlier

Per garantire l'affidabilità del dataset e migliorare le prestazioni del modello, sono state applicate diverse tecniche per gestire i valori **null** e **outlier**.

Gestione valori null

I valori nulli compromettono l'accuratezza del modello e portare ad eventuali errori durante la fase di training. Per questo motivo ho eseguito i seguenti step:

1. **Eliminazione delle righe con valori nulli in feature essenziali**
   - Se una feature importante per la predizione (es. **dati storici**, **chilometraggio totale**, ecc..) conteneva valori nulli, la riga viene rimossa dal dataset per evitare problemi durante il training.
2. **Sostituzione dei valori nulli con statistiche aggregative**
   - Per alcune metriche, invece di eliminare la riga, è stata applicata una sostituzione con la media dei valori disponibili per quello stesso veicolo. Per esempio se il valore della temperatura olio motore, risultava nullo per qualche giornata, è stato sostituito con la media dello stesso veicolo dei giorni precedenti.

Gestione degli outlier

Gli outlier sono stati individuati ed eliminati per evittare che valori estremi introducessero rumore nel modello e che lo influenzassero negativamente. Il metodo adottato è stato il **metodo dell'Interquartile Range (IQR)**:

1. **Calcolo dei quartili Q1 e Q3 per ogni feature numerica**
   - Q1 = primo quartile (25° percentile)
   - Q3 = terzo quartile (75° percentile)
   - IQR = Q3 - Q1
2. **Definizione delle soglie per gli outlier**
   - Sono stati considerati **outlier estremi** i valori al di fuori dell'intervallo: $[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]$

3. **Filtraggio dei dati**

    - Le righe con i valori al di fuori di questo range sono state rimosse dal dataset per garantire una distribuzione più omogenea.

4.1.5 Feature Engineering

Per migliorare la capacità predittiva del modello, sono state ingegnerizzate nuove feature basate su metriche telemetriche e informazioni storiche, con l'obiettivo di avere dati più rappresentativi. Il **feature engineering**, è il processo attraverso dei dati "grezzi" vengono trasformati in features che rappresentino meglio il problema che si vuole risolvere. Quest'ultimo è fondamentale nel contesto del Machine Learning perchè migliora di molto la qualità dei dati a nostra disposizione [10].

Aggregazione dei dati telemetrici

Le metriche telemetriche sono state aggregate a livello giornaliero per evitare un'eccessiva granularità e ridurre il rumore nei dati. In particolare sono state calcolate:

- **Media giornaliera** di ogni metrica per ottenere un valore rappresentativo della giornata.
- **Massimo giornaliero** di ogni metrica per catturare eventuali picchi anomali.

Creazione della feature "km_to_next_service"

Per ogni componente del veicolo, presente nello storico delle riparazioni, è stato calcolato il numero di chilometri rimanenti prima della successiva manutenzione, utilizzando una finestra temporale basata sull'ID veicolo e ID parte. L'operazione è stata implementata con la seguente logica:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import lead, col

window = Window.partitionBy("vehicleID", "partID").orderBy("km")

df = df.withColumn("next_km_service", lead("km").over(window))
df = df.withColumn("km_to_next_service",
                   col("next_km_service") - col("current_mileage")

# Se il valore risultante è negativo o nullo, viene sostituito con 0
df = df.withColumn("km_to_next_service",
                   when(col("km_to_next_service") < 0, 0)
```

```
13                .otherwise(col("km_to_next_service")))
```

Questa feature è fondamentale perchè permette al modello di apprendere il chilometraggio rimanente prima della prossima manutenzione.

Classificazione della frequenza di sostituzione

Per distinguere i diversi pattern delle manutenzioni e migliorare il modello, ho introdotto la feature "**freq_sostitution**", che classifica le sostituzioni in tre categorie:

- **"Frequent"**: sostituzioni avvenute con meno di 50.000 km percorsi.
- **"Medium"**: sostituzioni tra 50.000 e 200.000 km.
- **"Rare"**: sostituzioni oltre i 200.000 km.

L'operazione di assegnazione è stata implementata con la seguente logica:

```
from pyspark.sql.functions import when

df = df.withColumn("freq_sostitution",
                   when(col("km_to_next_service") < 50000, "frequent")
                   .when(col("km_to_next_service") < 200000, "medium")
                   .otherwise("rare"))
```

Bilanciamento del dataset

Siccome il dataset iniziale era sbilanciato, con una prevalenza di veicoli con sostiuzioni frequenti, è stato applicato un **undersampling** sulla classe "frequent", riducendola al 50% del suo volume originale per evitare che il modello fosse influenzato da questo squilibrio nei dati.

```
df_frequent = df.filter(col("freq_sostitution") == "frequent")
                        .sample(fraction=0.5, seed=42)
df_medium = df.filter(col("freq_sostitution") == "medium")
df_rare = df.filter(col("freq_sostitution") == "rare")

df = df_frequent.union(df_medium).union(df_rare)
```

L'obiettivo di questo bilanciamento è quello di garantire che il modello non impari a favorire le sostituzioni frequenti.

4.1.6 Selezione delle feature più rilevanti

Per migliorare le prestazioni del modello e ridurre la dimensione del dataset, è stata fatta una selezione delle fature più influenti per la manutenzione predittiva. Questo processo ha permesso di eliminare variabili ridondanti o poco significative, riducendo il rischio di un possibile overfitting.

Criteri di selezione delle feature

Sono stati adottati diversi approcci per determinare le feature da includere nel modello:

1. **Analisi della correlazione**
    - Ho calcolato la **correlazione di Pearson** tra ogni feature numerica e il targer "km_to_next_service".
    - Le feature con una correlazione prossima allo 0 sono state considerate meno influenti e rimosse.
    - Le feature con una correlazione molto alta tra loro, invece, sono state analizzate per mantenere solo quelle più rappresentative per il nostro caso di studio.
2. **Feature Importance tramite modelli preliminari**
    - Sono stati addestrati modelli di Machine Learnging preliminari (Random Forest e XGBoost) per calcolare l'importanza delle feature tramite il peso assegnato nei modelli agli alberi decisionali.
3. **Rimozione delle feature ridondanti o poco significative**
    - Le feature ridondanti o con una varianza troppo bassa, sono state eliminate.

Feature selezionate

Sono state selezionate le seguenti feature per l'addestramento del modello:

- **Metriche telemetriche aggregate**:
    - Temperatura olio motore media
    - Temperatura olio motore massima
    - Temperatura liquido refrigerante motore media
    - Temperatura liquido refrigerante motore massima

  - Pressione olio media
  - Pressione olio massima
  - Giri del motore medi
  - Giri del motore massimi
  - Chilometraggio totale attuale
- **Dati storici sulle manutenzioni**:
  - Km alla manutenzione
  - Giorno manutenzione
  - Mese manutenzione
  - Anno manutenzione
  - Identificativo parte
- **Dati caratteristici del veicolo**:
  - Identificativo del modello
  - Tipo di motorizzazione

Visualizzazione della Feature Importance

Per validare la selezione fatta, ho generato un grafico con l'importanza delle feature utilizzando un modello di Random Forest:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

importance = rf.feature_importances_
feature_names = X_train.columns
sorted_idx = importance.argsort()

plt.figure(figsize=(10, 6))
sns.barplot(x=importance[sorted_idx], y=feature_names[sorted_idx])
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Importanza delle Feature secondo Random Forest")
plt.show()
```

Questa analisi ha confermato l'importanza delle feature telemetriche e storiche delle manutenzioni per la predizione sulla prossima manutenzione.

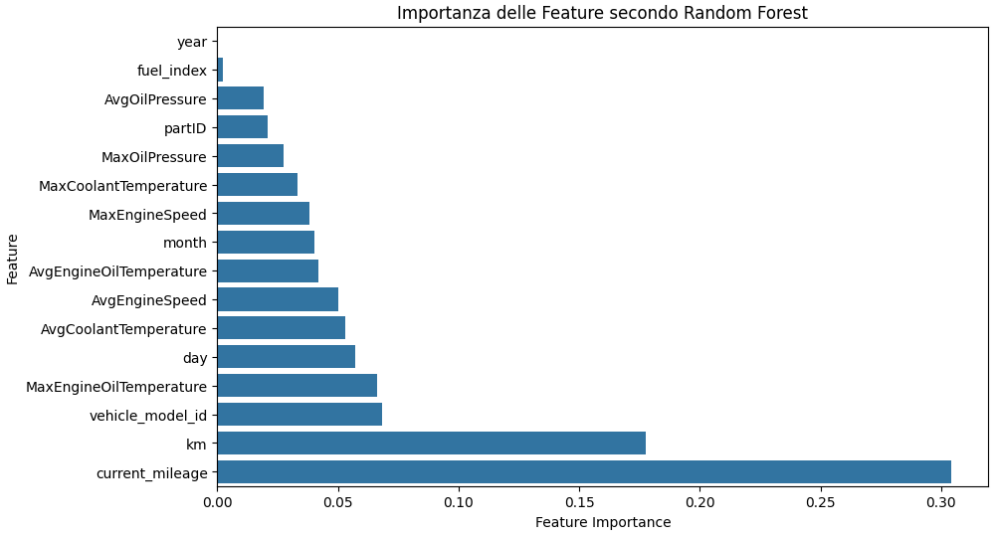

Figura 4.3: Feature Importance

4.1.7 Normalizzazione e trasformazione dei dati

I dati sono poi stati sottoposti ad una fase di **normalizzazione** e di **trasformazione**.

Feature Scaling per la gestione degli outlier

Siccome il dataset contiene feature numeriche con scale diverse e possibili outlier, è stato utilizzato il **Robust Scaler**, che riduce l'impatto dei valori estremi normalizzando i dati rispetto alla mediana e all'intervallo interquantile (IQR). L'implementazione è la seguente:

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Ho preferito il **Robust Scaler** rispetto ad altri scaler, come per esempio MinMaxScaler o StandardScaler perchè meno sensibile agli outlier, garantendo una trasformazione più stabile.

Conversione delle feature per Spark MLlib

Per l'integrazione con **Spark MLlib**, tutte le feature sono state convertite in un vettore di feature utilizzando la funzione "**VectorAssemble**" di PySpark:

```
from pyspark.ml.feature import VectorAssembler

feature_cols = ["f1", "f2", "f3", ...]  # Lista delle feature
a = VectorAssembler(inputCols=feature_cols, outputCol="features")
df = a.transform(df)
```

Questa trasformazione permette di avere i dati in un formato compatibile.

Capitolo 5

Sviluppo del Modello Predittivo

In questo capitolo, verrà descritto come è stato sviluppato il modello e quali risultati sono riuscito ad ottenere, facendo poi un'analisi dei grafici e delle metriche ottenute.

5.1 Definizione del problema e scelta del target

L'obiettivo del modello è prevedere il numero di chilometri rimanenti prima della prossima manutenzione per ciascuna parte del veicolo, utilizzando sia i dati storici che quelli telemetrici. Il target scelto per la previsione è:

$$km\_to\_next\_service = next\_km\_service - current\_mileage \tag{5.1}$$

dove:

- **next_km_service** è il chilometraggio in cui deve avvenire la prossima manutenzione.
- **current_mileage** è il chilometraggio attuale del veicolo.

Questo consente di stimare in modo dinamico quando sarà necessaria una manutenzione, milgiorando la gestione delle flotte di veicoli.

5.2 Modelli di Machine Learning testati

Per identificare il modello più efficace, ho testato diversi algoritmi di regressione, valutandoli in termini di **accuratezza**, **interpretabilità** e **scalabilità** su grandi dataset.

5.2.1 Random Forest

Un insieme di alberi decisionali che fonde i risultati di ciascun albero per migliorare la robustezza delle previsioni.

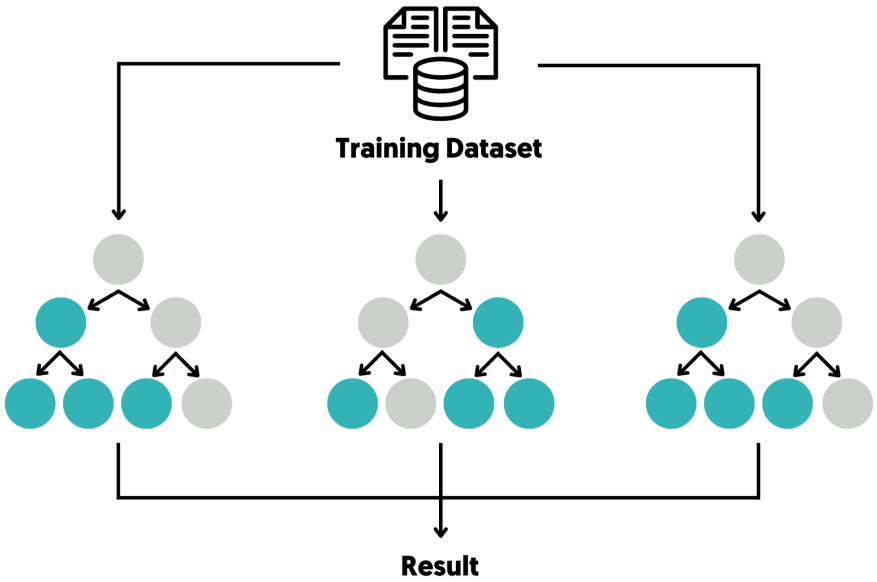

Figura 5.1: Random Forest [11]

```
from pyspark.ml.regression import RandomForestRegressor

rf = RandomForestRegressor(featuresCol="features", labelCol="
    km_to_next_service", numTrees=100)
rf_model = rf.fit(train_df)
```

Pro	Contro
Buona interpretabilità	Richiede molto tempo per il training
Gestisce bene le feature non lineari	-

Tabella 5.1: Pro e Contro di Random Forest

5.2.2 Gradient Boosting (XGBoost)

Algoritmo basato sull'uso di alberi devisionali e un gradiente, che migliora iterativamente la previsione.

Figura 5.2: XGBoost [12]

```
from xgboost.spark import SparkXGBRegressor

xgb = SparkXGBRegressor(features_col="features", label_col="
    km_to_next_service", num_round=100)
xgb_model = xgb.fit(train_df)
```

Pro	Contro
Alta accuratezza	Meno interpretabile rispetto a Random Forest
Buona gestione degli outlier	-

Tabella 5.2: Pro e Contro di XGBoost

5.2.3 Light Gradient-Boosting Machine (LightGBM)

Un'alternativa a XGBoost, ottimizzata per grandi dataset.

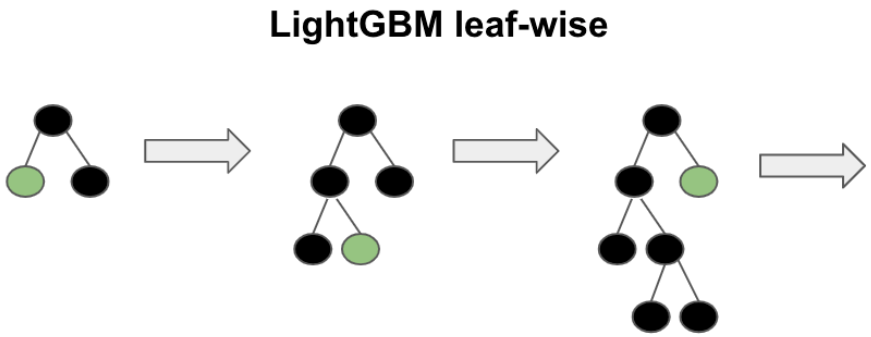

Figura 5.3: Light-GBM [13]

```
from synapse.ml.lightgbm import LightGBMRegressor

lgb = LightGBMRegressor(featuresCol="features", labelCol="
    km_to_next_service", numLeaves=31, numIterations=100)
lgb_model = lgb.fit(train_df)
```

Pro	Contro
Più veloce di XGBoost	Meno robusto agli outlier
Minor consumo di memoria	-

Tabella 5.3: Pro e Contro di LightGBM

5.2.4 CatBoost

Boosting categorico per feature categoriche.

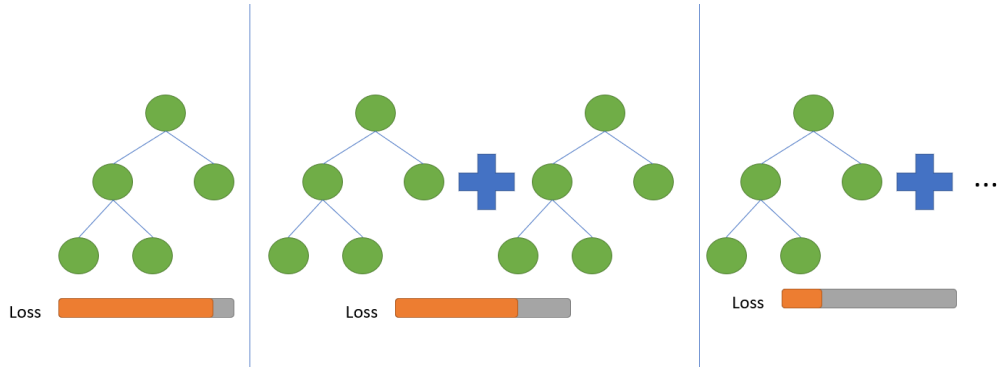

Figura 5.4: CatBoost [14]

```
from catboost import CatBoostRegressor

cat_model = CatBoostRegressor(iterations=100, depth=6, learning_rate
    =0.1)
cat_model.fit(X_train, y_train)
```

Pro	Contro
Buone prestazioni anche senza normalizzazione	Training lento
Ottimizzato per dati tabellari	-

Tabella 5.4: Pro e Contro di CatBoost

5.3 Addestramento dei modelli con Spark MLlib

I modelli sono stati addestrati su **Google Colab**, sfruttando la scalabilità di Spark per gestire il grande volume di dati di cui disponevo. Il dataset è poi stato trasformato in uun formato compatibile con Spark MLlib prima del training:

```
from pyspark.ml.feature import VectorAssembler

feature_cols = ["f1", "f2", "f3", ...]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="
    features")
df = assembler.transform(df)
```

5.4 Metriche di valutazione

Per valutare le prestazioni dei modelli, sono state utilizzate due metriche di regressione:

- **Root Mean Squared Error (RMSE)**
- **Mean Absolute Error (MAE)**

Queste sono state calcolate in PySpark:

```
from sklearn.metrics import mean_absolute_error

# Compute RMSE
rmse_rf = np.sqrt(mean_squared_error(y_test, rf_preds))
rmse_lgbm = np.sqrt(mean_squared_error(y_test, lgbm_preds))
rmse_cb = np.sqrt(mean_squared_error(y_test, catboost_preds))
rmse_xgb = np.sqrt(mean_squared_error(y_test, xgboost_preds))
rmse_meta = np.sqrt(mean_squared_error(y_test, meta_preds))

# Compute MAE
mae_rf = mean_absolute_error(y_test, rf_preds)
mae_lgbm = mean_absolute_error(y_test, lgbm_preds)
mae_cb = mean_absolute_error(y_test, catboost_preds)
mae_xgb = mean_absolute_error(y_test, xgboost_preds)
mae_meta = mean_absolute_error(y_test, meta_preds)
```

5.4.1 RMSE

L'RMSE, anche conosciuto come l'errore quadratico medio, indica la discrepanza quadratica media fra i valori dei dati osservati e quelli dei dati stimati. Non è una quantità a-dimensionale, ma assume l'unità di misura della grandezza considerata[15], quindi in questo caso i chilometri (km). Si calcola come:

$$RMSE = \sqrt{\frac{1}{N}\sum_{(i=1)}^{N}(y_i - \hat{y}_i)^2} \tag{5.2}$$

5.4.2 MAE

Il MAE, anche conosciuto come errore medio assoluto, misura la media degli errori assoluti tra i valori previsti dal modello ed i valori osservati. Anch'esso non è

a-dimensionale ma assume l'unità di misura della grandezza considerata (km). Si calcola come:

$$MAE = \frac{1}{N} \sum_{(i=1)}^{N} |y_i - \hat{y}_i| \tag{5.3}$$

5.4.3 Confronto tra modelli

Confrontando i vari modelli otteniamo:

Modello	RMSE (km)	MAE (km)
Random Forest	836.19	437.36
XGBoost	620.34	311.24
LightGBM	643.80	365.00
CatBoost	679.16	386.22
Meta-Modello (XGBoost)	11.86	7.43

Tabella 5.5: Confronto tra le metriche dei vari modelli utilizzati

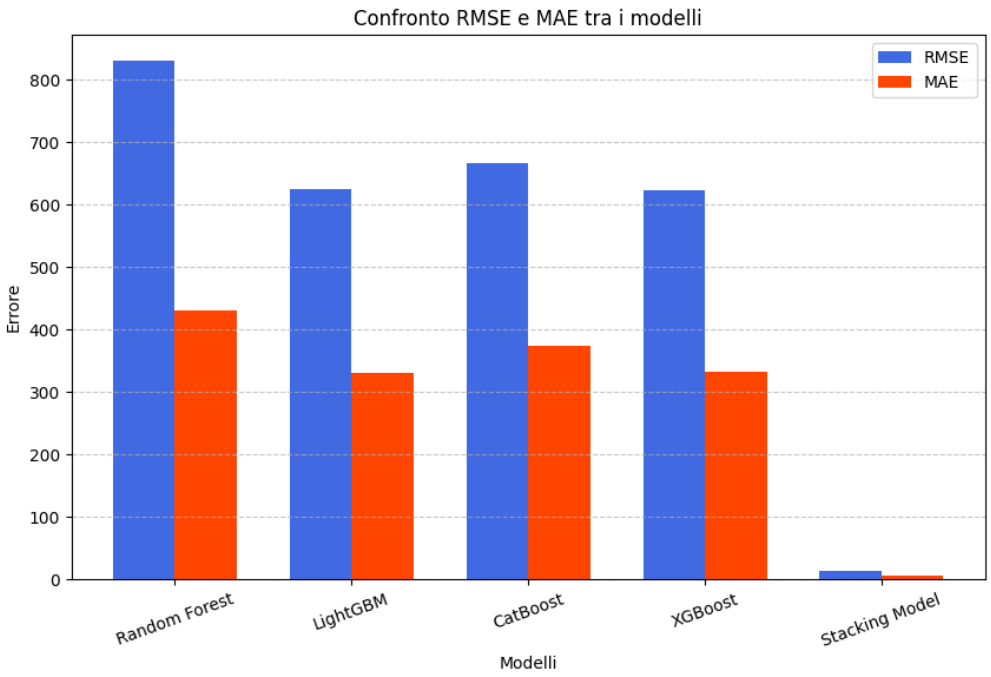

Figura 5.5: Confronto RMSE e MAE tra modelli

Possiamo notare che:

- **XGBoost** ha il miglior valore di RMSE e di MAE, risultando il modello più accurato, anche per questo ho scelto di usarlo come meta-modello.
- **LightGBM** ha prestazioni molto simili, risultando un po' più rapido in fase di training.
- **Random Forest** è poco preciso rispetto agli altri.
- **CatBoost** ottiene prestazioni di poco peggiori, ma risulta molto lento nel training.

XGBoost rappresenta il miglior trade-off tra prestazioni e accuratezza, per questo è stato scelto anche come meta-modello.

Error Distribution - Meta Model

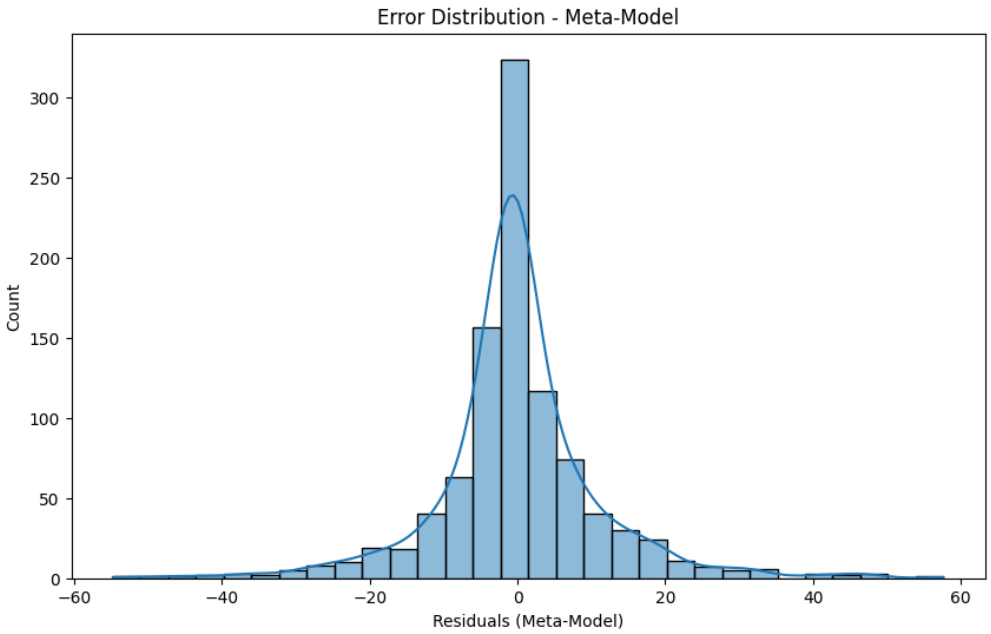

Figura 5.6: Error Distribution del meta-modello

- La distribuzione dei residui è centrata intorno allo zero e simmetrica, questo è un buon segno. Indica che il modello non ha un bias sistematico e che gli errori sono bilanciati.
- La maggior parte dei residui è concentrata vicino allo zero, significa che il modello ha una buona precisione.

Residual Plot - Meta Model

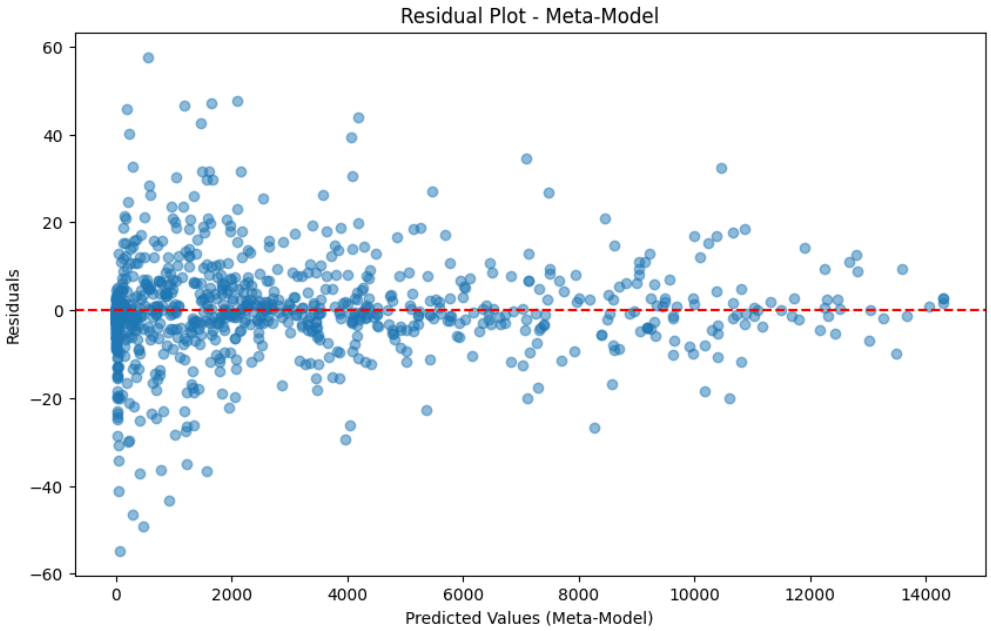

Figura 5.7: Residual Plot del meta-modello

- I residui sono sparsi casualmente intorno allo zero senza mostrare pattern evidenti (come curve o dispersione crescente), questo significa che il modello sta catturando bene le relazioni nei dati.

Residual Analysis - Models

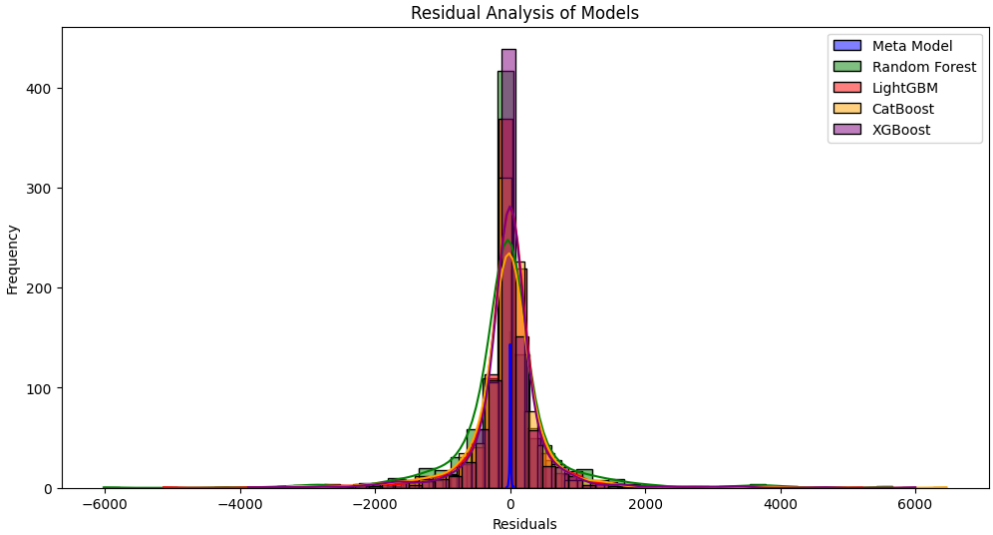

Figura 5.8: Residual Analysis dei modelli

- I residui del meta-modello sono più vicini allo zero rispetto a quelli degli altri modelli, questo indica che il meta-modello è il più accurato.

Capitolo 6

Deploy e Scalabilità del Sistema

In questo capitolo vengono descritti i passi che ho seguito per il deploy e per la scalabilità del sistema di manutenzione predittiva.

6.1 Pipeline di elaborazione dei dati

L'elaborazione dei dati è un passaggio fondamentale per andare a creare il dataset finale su cui fare addestramento del modello predittivo. La pipeline si occupa di raccogliere, trasformare e caricare i dati ricevuti in un formato compatibile con il sistema. I dati provengono da due fonti principali:

- **Database Veicoli**: contiene informazioni storiche sui veicoli, inlcusi tutta la cronologia delle manutenzioni.
- **Database Geotab**: contiene le informazioni delle metriche in tempo reale sui veicoli, come posizione, temperature interne del motore, giri del motore ed altre metriche.

6.1.1 Caricamento dei dati

1. **Recupero e aggiornamento dei dati dal database Veicoli**
    - I dati vengono estratti dal database che contiene le informazioni "anagrafiche" di ogni veicolo e anche la sua storia delle manutenzioni.
    - Per mantenere i dati aggiornati, viene fatta una chiamata API periodica per avere sempre le informazioni sincronizzate.

2. **Acquisizione delle metriche dal database GeoTab**

    - Le metriche dei veicoli, come chilometraggio, giri del motore, temperatura del motore, ecc.. vengono raccolte ed elaborate dal sistema GeoTab.
    - Per mantenere questi dati aggiornati, viene fatta una chiamata API giornaliera che interroga il database Geotab e memorizza le informazioni nel sistema.
    - I dati ricevuti vengono poi elaborati e utilizzati insieme ai dati storici.

Durante tutta questa fase vengono applicate numerose tecniche di **data cleaning** e di **normalizzazione** per avere dati coerenti e utilizzabili nel modello predittivo.

6.1.2 Integrazione con il back-end Nest.js per l'elaborazione dei dati

Il back-end Nest.js ha il compito di orchestrare l'elaborazione dei dati e la gestione dei dati ricevuti, senza gravare sulla UI. Le sue funzionalità principali sono:

- **Gestione delle chiamate API**
    - API per interrogare il database veicoli e aggiornare i dati storici.
    - API per connettersi al database GeoTab del cliente selezionato e recuperare le metriche giornaliere dei veicoli.
- **Pre-elaborazione dei dati per il modello predittivo**
    - Conversione dei dati grezzi in feature utilizzabili dal modello predittivo.
    - Aggregazione e filtraggio dati per rimuovere informazioni mancanti o ridondanti.
    - Calcolo di metriche derivate, come le medie delle temperature e anche il massimo di quest'ultime.

6.2 Implementazione della Dashboard di Monitoraggio

La dashboard di monitoraggio è un'interfaccia utente progettata per essere chiara e facilmente utilizzabile ed interpretabile, e sapere lo stato attuale di ogni veicolo della propria flotta.

6.2.1 Strutturazione della UI con Vue.js

Per lo sviluppo dell'intera UI è stato scelto Vue.js in combinazione con PrimeVue, una libreria di componenti UI che offre un set di elementi grafici predefiniti. L'uso di quest'ultimo semplifica la costruzione di un'interfaccia moderna e reattiva, migliorando l'esperienza utente. L'architettura del sistema è strutturato per offrire una navigazione chiara e un accesso immediato ai dati di interesse. Le sezioni principali della webapp sono:

- **Pannello di riepilogo**: in cui vengono visualizzate alcune metriche chiave, come il numero di veicoli monitorati, il fornitore gps, l'accuratezza della predizione e la tabella in cui si vedono le predizioni per ogni parte dell'intera flotta.
- **Sezione di dettaglio flotta**: in cui viene visualizzata una tabella con tutti i dati della predizione per ogni veicolo della flotta.
- **Sezione di dettaglio veicolo**: in cui viene visualizzato il veicolo singolo, con i dati sia in forma tabellare che tramite grafico.

6.2.2 Visualizzazione dei dati

Per rappresentare i dati ho utilizzato i componenti di PrimeVue. In particolare abbiamo:

- **Grafici interattivi**: ho usato il componente Chart di PrimeVue, grazie a questo visualizzo grafici a barre che indicano la vita della parte. I dati di quest'ultimo vengono aggiornati tramite API REST.
- **Tabella dati**: per mostrare la predizione per ogni parte sull'intera flotta, ho utilizzato il componente DataTable, con uno specifico ordinamento (crescente) per mostrare subito all'utente le parti con uno stato di usura più critico.
- **Card dinamiche**: utilizzate principalmente nella homepage per rendere la UI un po' più accattivante, pur dando informazioni utili all'utilizzatore.

6.2.3 Integrazione con il back-end Nest.js per il recupero dei dati predittivi

La dashboard comunica con il back-end Nest.js tramite **API REST**, recuperando i dati delle elaborazioni richieste. L'integrazione è stata fatta per avere un architettura asincrona, in modo da garantire un continuo flusso di informazioni tra sistema e utente. Le principali operazioni di integrazione hanno interessato:

- **Recupero delle previsioni di manutenzione**: il front-end fa richiesta API giornalmente per ottenere i dati aggiornati per ogni parte dell'intera flotta veicoli e per ottenere il chilometraggio previsto di vita.

- **Aggiornamento automatico dei dati**: i dati vengono aggiornati automaticamente all'inizio di ogni giorno, perchè le metriche telemetriche sono calcolate giornalmente.

- **Gestione delle notifiche**: il front-end riceve dal back-end delle notifche che vengono mostrate all'utente mediante dei Toast.

6.2.4 Sistema di cache e ottimizzazione del caricamento

Per rendere l'esperienza utente meno frustrante, migliorare le prestazioni e ridurre il carico sul server e sulle sue API, la dashboard utilizza **localStorage** come sistema di caching interno. Questo meccanismo mi consente di:

- **Ridurre le chiamate API**, i dati critici vengono salvati in cache perchè una seconda elaborazione durante la stessa giornata sarebbe inutile e molto pesante. La cache viene poi invalidata parzialmente alla fine di ogni giornata.

- **Migliorare la velocità di caricamento**, quando l'utente accede alla dashboard i dati vengono caricati immediatamente da localStorage.

- **Strategia di aggiornamento parziale**, i dati storici e l'autenticazione viene mantenuta in cache più a lungo rispetto ai dati telemetrici, che invece vengono eliminati giornalmente per fare spazio a quelli nuovi ed aggiornati.

6.2.5 Schermate Applicazione

Le seguenti schermate sono un prototipo fedele di ciò che è l'applicazione, sono stati selezionati solo alcuni veicoli per rendere le schermate più chiare e intuitive da vedere.

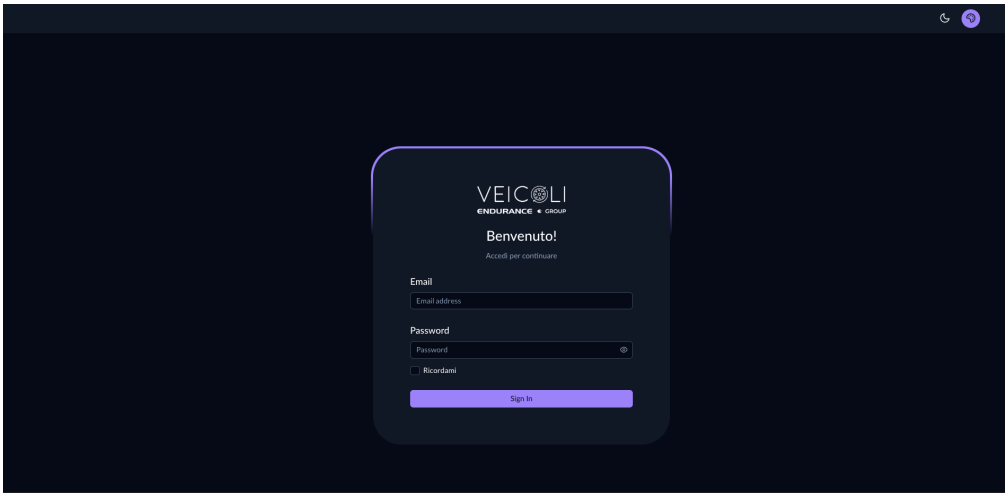

Figura 6.1: Schermata Login del sistema

La schermata di Login è composta da un semplice form di accesso.

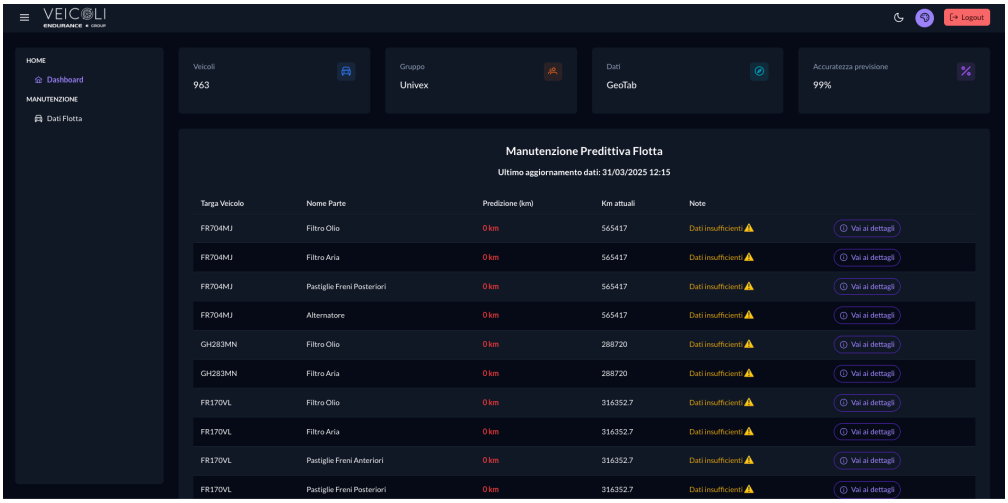

Figura 6.2: Schermata Home del sistema

La seguente schermata rappresenta la Home e mostra le card dinamiche, il menù laterale e anche la tabella con le predizione di ogni parte dell'intera flotta. Nelle note compare "Dati Insufficienti" quando mancano alcune metriche che possono influenzare negativamente quella predizione.

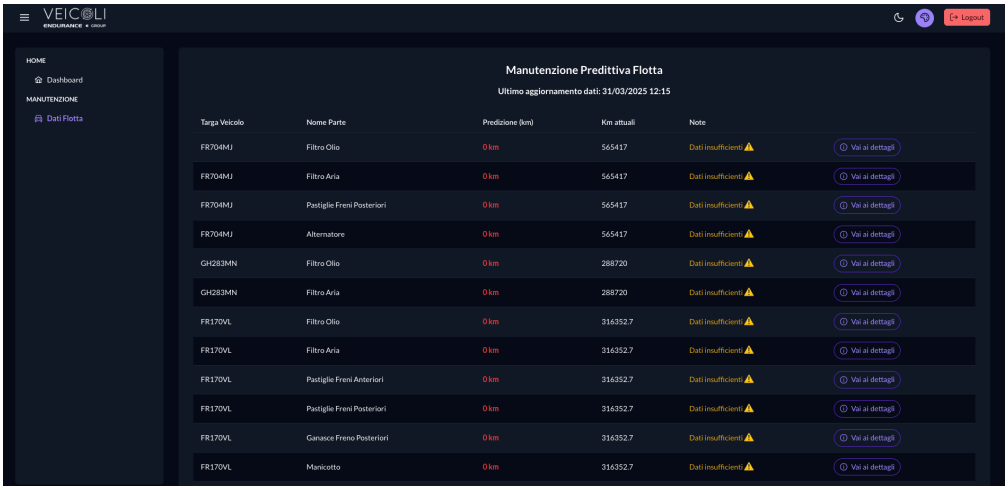

Figura 6.3: Schermata Dettaglio Flotta del sistema

La seguente schermata rappresenta solamente la tabella con la predizione per ogni parte dell'intera flotta. Come prima, nelle note compare "Dati Insufficienti" quando mancano alcune metriche che possono influenzare negativamente quella predizione.

Figura 6.4: Schermata Dettaglio Veicolo del sistema

Figura 6.5: Schermata Dettaglio Veicolo del sistema (parte 2)

Le seguenti immagini rappresentano il dettaglio di un singolo veicolo, selezionato nella tabella precedente. I dati vengono mostrati sia tramite tabella che tramite grafico, inoltre se alcune metriche in quella giornata mancano, vengono visualizzate in modo che l'utente può sapere ciò che manca e verificare eventuali problemi.

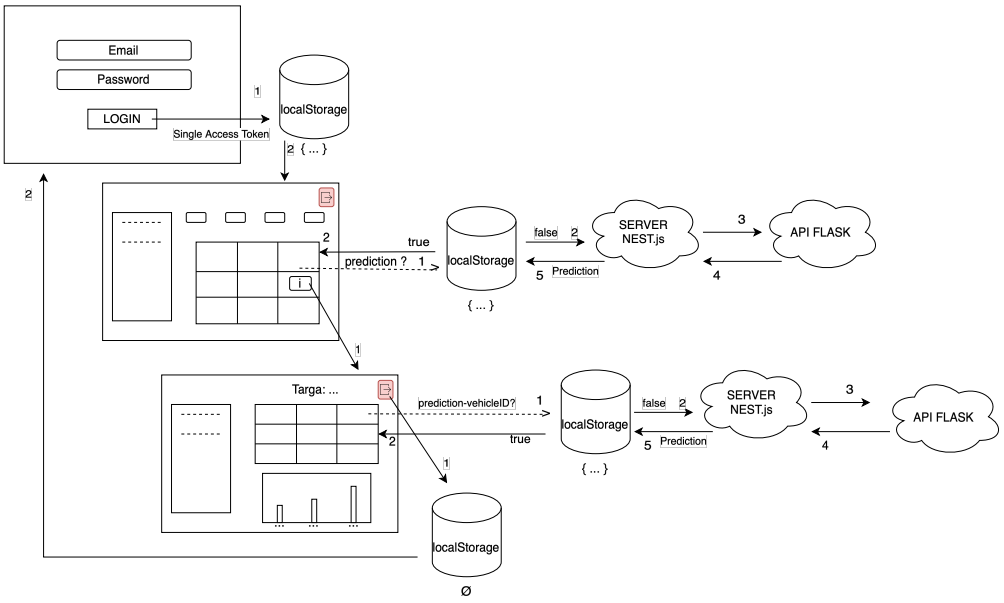

Figura 6.6: Logica del sistema

Quando viene fatto il login, viene salvato nella cache (localStorae) il **Single Access Token** dell'utente, si viene poi dirottati verso la home page in cui i

dati presenti in tabella sono presi dalla cache se presenti ed aggiornati al giorno odierno, oppure presi tramite una richiesta al server Nest.js, il quale a sua volta contatta l'API con il modello. Una volta ricevuta la predizione, questa viene salvata nella cache per rendere una seconda renderizzazione, più veloce. Quando si vuole visualizzare un veicolo in particolare, viene fatto lo stesso controllo fatto precedentemente e poi si rendono disponibili i dati. Infine, se viene fatto il logout, la cache si cancella totalmente, riportado l'utente alla pagina di login. La cache dei dati predittivi, viene svuotata giornalmente per fare spazio ai nuovi dati aggiornati.

6.3 Containerizzazione e Deploy su Google Cloud Run

Per garantire la **scalabilità**, l'**affidabilità** e la **semplicità** di deploy, l'intero sistema è stato containerizzato con Docker e distrubuito tramite Google Cloud Run, un servizio serverless che permette l'esecuzione di container caricati in appositi registri interni.

6.3.1 Creazione dei container Docker per API modello, back-end Nest.js e UI Vue.js

Il sistema è stato suddiviso in tre container Docker distinti, ognuno con un ruolo specifico:

1. **API del modello**
    - Esegue le previsioni di manutenzione utilizzando i dati forniti.
    - Riceve richieste HTTP e restituisce i risultati in formato JSON.
    - Espone un'API per l'integrazione con il back-end Nest.js.
2. **Back-end Nest.js**
    - Gestisce le richieste dell'utente e elabora i dati sia predittivi che storici.
    - Comunica con i database e con l'API del modello.
    - Autentica gli utenti.
3. **UI Vue.js**
    - Fornisce l'interfaccia grafica agli utenti.
    - Recupera i dati dal back-end e li visualizza.
    - Utilizza componenti reattivi di PrimeVue.

- Gestisce la cache per le richieste più frequenti e pesanti.

Questo è il Dockerfile utilizzato per creare il container dell'API del modello:

```
# Usa un'immagine base Python
FROM python:3.9

# Imposta la working directory
WORKDIR /app

# Copia i file necessari
COPY requirements.txt .

# Installa le dipendenze
RUN pip install --no-cache-dir -r requirements.txt

# Copia il codice dell'API
COPY . .

# Espone la porta 5000 per la comunicazione
EXPOSE 5000

# Avvia l'applicazione
CMD ["python", "app.py"]
```

6.3.2 Deploy e gestione della scalabilità con Google Cloud Run

Google Cloud Run è stato scelto per i suoi vantaggi serverless quali:

- **Scalabilità automatica**, crea o elimina istanze in base al traffico.
- **Gestione semplificata**, non richiede la gestione di un server.
- **Costo ottimizzato**, si paga solo quando viene effettivamente eseguito uno dei servizi.

Deploy su Google Cloud Run

Dopo aver costruito e pushato le immagini Docker su Google Artifact Registry, il deploy su Google Cloud Run viene eseguito con:

```
gcloud run deploy api-predictive-maintenance \
```

```
2 --image europe-docker.pkg.dev/predictive-maintenance-veicoli/
    predictive-maintenance/api-predictive-maintenance:v1 \
3 --region europe-west1 \
4 --platform managed \
5 --allow-unauthenticated \
6 --memory=2.5Gi
```

Lo stesso processo viene ripetuto per il back-end Nest.js e per la UI Vue.js.

Capitolo 7

Risultati e Analisi

In questo capitolo verranno analizzati i risultati ottenuti dal sistema di manutenzione predittiva, andando ad analizzare le prestazioni del modello, i vantaggi operativi introdotti e alle sfide affrontate durante lo sviluppo del sistema.

7.1 Prestazioni del modello predittivo

L'efficacia e le prestazioni del modello predittivo sono state valutate attraverso metriche di errore e accuratezza, analizzando la capacità del sistema di prevedere il momento giusto per sostituire le parti dei veicoli.

7.1.1 Metriche di Valutazione

Il modello è stato testato su un dataset di dati diviso in questo modo:

```
# Split the dataset in train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
```

Ottenendo che il test set è il 20% dell'intero dataset, mentre il restate 80% sono dati destinati all'addrestramento. I risultati ottenuti dopo lo stacking, ossia l'uso di un meta-modello, sono stati i seguenti: Questo significa che la predizione può

Metrica	Valore ottenuto
RMSE (Root Mean Squared Error)	11.86 km
MAE (Mean Absolute Error)	7.43 km

Tabella 7.1: Performance del meta-modello

avere un incertezza, in positivo o in negativo, di 7.43km, il quale è un valore ottimo per applicazioni reali.

7.1.2 Analisi delle Previsioni

Le previsioni ottenute, sono state confrontate con i dati reali di manutenzione, ed è stato possibile notare come si ha un'alta precisione per i componenti soggetti ad usura graduale (es. pastiglie dei freni, filtri olio, filto aria, ecc.) rispetto a componenti con usura più variabile (es. batterie, sensori elettronici, ecc.). Per migliorare queste previsioni, bisognerebbe integrare anche dati ambientali e condizioni di guida per stimare al meglio l'usura delle parti critiche.

7.2 Vantaggi operativi e impatto aziendale

L'introduzione della manutenzione predittiva ha portato diversi vantaggi operativi, migliorando efficienza, riducendo costi e aumentando la sicurezza delle flotte.

7.2.1 Riduzione dei Costi

La manutenzione predittiva consente di:

- Ridurre gli interventi di emergena, che hanno costi più elevati rispetto alla manutenzione ordinaria.
- Ottimizzare la gestione dei ricambi, evitando scorte eccessive o carenze improvvise di parti meccaniche.
- Diminuire il fermo macchina, aumentando la disponibilità dei veicoli e la loro efficienza.

7.2.2 Miglioramento dell'Operatività

La manutenzione predittiva ha anche migliorato l'organizzazione delle operazioni aziendali:

- Programmazione efficiente degli interventi, evitando sovraccarichi di lavoro nei centri di assistenza.
- Monitoraggio in tempo reale dello stato del veicolo.
- Maggior affidabilità della flotta, riducendo di molto la probabilità di guasti improvvisi.

7.2.3 Benefici in termini di Sicurezza

La manutenzione predittiva ha contribuito a:

- Ridurre il rischio di guasti improvvisi su componenti critici.
- Migliorare la sicurezza degli autisti, riducendo la probabilità di incidenti dovuti a guasti tecnici.
- Garantire conformità alle norme.

7.3 Limiti e sfide del lavoro svolto

Nonostante i risultati positivi ottenuti, lo sviluppo del sistema ha presentato alcune sfide di natura tecnica e operativa.

7.3.1 Limitazioni del Modello

- **Difficoltà nel prevedere guasti improvvisi**, il modello si basa sui dati storici, quindi su tendenze di usura, non riesce a prevedere eventi casuali come guasti elettronici o danni accidentali.
- **Mancanza di dati ambientali**, le condizioni ambientali possono influenzare l'usura delle parti, ma questi dati sono difficili da ricavare e da utilizzare in modo efficace.
- **Dipendenza dalla qualità dei dati**, se i dati storici sono incompleti o imprecisi, le previsioni potrebbero risultare meno affidabili.

7.3.2 Sfide Tecniche

- **Gestione di grandi moli di dati**, l'integrazione dei dati provenienti dai sensori, database storici e API ha richiesto una fase corposa di ottimizzazione delle query e di caching per garantire delle buone prestazioni.
- **Sincronizzazione tra back-end e front-end**, l'aggiornamento in tempo reale dei dati ha richiesto una gestione ottimale della comunicazione tra i due componenti, andando a sfruttare il sistema del caching con localStorage per ridurre il carico sul server.
- **Scalabilità del sistema**, utilizzare Google Cloud garantisce una buona scalabilità, ma comunque le prestazioni sono ancora lontane da essere ottime e potrebbero essere necessarie ulteriori ottimizzazioni per rendere il sistema migliore.

Capitolo 8

Conclusioni e Sviluppi Futuri

In questo capito vengono riassunti i principali risultati della ricerca, evidenziando i risultati ottenuti e le potenziali evoluzioni future.

8.1 Contributi della ricerca

Il lavoro ha portato allo sviluppo di un sistema di manutenzione predittiva, che combina dati storici e telemetrici per prevedere quando i veicoli necessitano di una manutenzione. Il modello predittivo, è basato su stacking ha dimostrato una buona precisione, specialmente per le componenti soggette a un'usura graduale. Il principale contributo è stato sicuramente l'integrazione dei dati provenienti dalle due fonti, creando un infrastruttura in grado di aggiornare continuamente le previsioni. Il server e la UI hanno rappresentato un valore aggiunto, permettendo di essere interpretabili e fruibili i dati all'utente finale. Dal punto di vista operativo, questo sistema ha migliorato l'efficienza della gestione della manutenzione, riducendo i guasti improvvisi e ottimizzando la flotta di veicoli. I vantaggi, sono stati evidenti in termini di riduzione dei costi, minori tempi di fermo macchina e maggiore affidabilità della flotta.

8.2 Possibili miglioramenti futuri

Nonostante i risultati soddisfacenti, ci sono ancora margini per migliorare la predizione. Un primo aspetto riguarda l'integrazione dei dati ambientali, come la temperatura, l'umidità, le condizioni stradali e il luogo in cui il veicolo opera, che potrebbero influenzare l'usura di alcuni componenti. L'aggiunta di queste

informazioni renderebbe il modello più preciso e più versatile, rendendolo capace di adattarsi a scenari diversi. Un'altra possibile evoluzione riguarda l'utilizzo di tecniche di **Anomaly Detection** per identificare guasti improvvisi che non seguono un patter di usura graduale. Utilizzare modelli basati sulle reti neurali potrebbe migliorare la capacità di rilevare comportamenti anomali nei dati e aumentare l'affidabilità delle previsioni. Per quanto riguarda il punto di vista tecnico, l'ottimizzazione dell'infrastruttura, in particolare quella del server, potrebbe contribuire a migliorare le prestazioni del sistema. Creando un'infrastruttura basata su microservizi, permetterebbe non solo una maggiore scalabilità, ma anche una gestione migliore delle richieste. Inoltre si potrebbe anche migliorare la gestione del caching e perfezionare le query sui database che potrebbero ridurre di molto i tempi di risposta e quindi migliorare l'esperienza utente. Infine, anche la dashboard potrebbe essere ampliata con altre funzionalità, come l'integrazione di notifiche in tempo reale per segnalare interventi imminenti che potrebbero migliorare l'efficacia del sistema.

8.3 Applicabilità in contesti industriali

Il sistema ha un grande potenziale a livello applicativo in diversi settori industriali. Nel settore logistico e dei trasporti, l'utilizzo di questo sistema permetterebbe di ridurre i guasti imprevisti e migliorare l'affidabilità delle consegne, contribuendo ad una migliore organizzazione delle flotte. Anche l'industria manifatturiera potrebbe trarre dei vantaggi utilizzando questo sistema, applicandolo ai macchinari e agli impianti di produzione per ridurre i fermi non programmati e migliorare la pianificazione degli interventi. Nel settore agricolo, questo approccio potrebbe essere utilizzato per la gestione delle macchine agricole e delle attrezzature adoperate, ottimizzando anche qui i tempi di utilizzo e prevedendo guasti. Questo sistema potrebbe anche portare benefici in termini di sostenibilità ambientale, un argomento molto sensibile negli ultimi tempi, perchè permetterebbe di ridurre gli sprechi di componenti e ottimizzerebbe le risorse. Una manutenzione più efficiente, porta ad una minore produzione di rifiuti meccanici e ad un utilizzo più responsabile dei materiali di consumo.

8.4 Conclusioni Finali

Il progetto sviluppato in questa tesi ha dimostrato come l'analisi avanzata dei dati a disposizione e l'uso di modelli di machine learning possano migliorare l'efficienza operativa, ridurre i costi e aumentare la sicurezza dell'intera flotta di veicoli. L'implementazione del sistema ha portato allo sviluppo di un modello predittivo in grado di stimare con una buona precisione il ciclo della vita delle

componenti soggette ad usura. Le metriche su cui il modello è stato valutato, hanno mostrato prestazioni soddisfacenti, dimostrando che il modello riesca ad anticipare correttamente la necessità di interventi, con un margine di errore molto contenuto. Tuttavia, l'analisi ha dimostrato che il modello non riesce a predire i guasti improvvisi e che la qualità dei dati influenza direttamente la predizione. Dal punto di vista applicativo, il sistema può essere integrato efficientemente nelle aziende e può essere sfruttato per portare benefici in termini di pianificazione e gestione delle risorse. Prevedere in anticipo il momento migliore per gli interventi, consente di minimizzare i tempi di fermo dei veicoli, migliorare la gestione dei ricambi e aumentare l'affidabilità dei veicoli della flotta. Durante lo sviluppo, ci sono state diverse sfide affrontate, come la gestione di grandi quantità di dati, la scalabilità del sistema e l'integrazione tra le varie tecnologie utilizzate. Per migliorare ancora di più i risultati ottenuti si possono aggiungere informazioni ambientali e utilizzare modelli basati su reti neurali. L'applicabilità di questo sistema non si limita solo al settore automotive, ma può essere esteso a tutti i settori che fanno uso di mezzi e veicoli che sono dotati di parti meccaniche che tendono ad usurarsi nel tempo. L'integrazione di questo sistema è agevolata dal continuo sviluppo delle ultime tecnologie, come l'IoT e l'intelligenza artificiale avanzata. Infine, possiamo dire che questo caso di studio rappresenta un passo importante verso la digitalizzazione delle manutenzioni e comporta anche un miglioramento interno nell'organizzazione. Inoltre, l'adozione di questo sistema potrà ridurre i costi, migliorare le prestazioni e contribuire a ridurre gli sprechi di risorse e quindi ad essere più sostenibili.

Bibliografia

[1] Gandalf Project. *Machine Learning Academy*. URL: https://gandalfproject.com/machine-learning-academy/ (cit. a p. 7).

[2] Michael Chen. *Cosa sono i Big Data?* URL: https://www.oracle.com/it/big-data/what-is-big-data/ (cit. a p. 8).

[3] SAP Business Technology Platform. *Cosa sono i Big Data?* URL: https://www.sap.com/italy/products/technology-platform/what-is-big-data.html (cit. a p. 8).

[4] SAP Business Technology Platform. *Che cos'è l'Internet of Things (IoT)?* URL: https://www.sap.com/italy/products/technology-platform/what-is-iot.html (cit. a p. 9).

[5] AWS. *Cos'è Apache Spark?* URL: https://aws.amazon.com/it/what-is/apache-spark/ (cit. a p. 15).

[6] Geotab Team. *Introducing the Geotab GO9+: Take Wi-Fi with you wherever you go*. URL: https://www.geotab.com/blog/geotab-go9-plus/ (cit. a p. 17).

[7] dremio. *Grid Search*. URL: https://www.dremio.com/wiki/grid-search/#:~:text=Grid%20Search%20is%20a%20traditional,the%20limitations%20of%20Grid%20Search%3F (cit. a p. 19).

[8] Keaun Amani. *Understanding Grid Search as an Optimization Algorithm in Machine Learning*. URL: https://neurosnap.ai/blog/post/understanding-grid-search-as-an-optimization-algorithm-in-machine-learning/643748be49872f3862f39aed (cit. a p. 20).

[9] Brijesh Soni. *Stacking to Improve Model Performance: A Comprehensive Guide on Ensemble Learning in Python*. URL: https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28 (cit. a p. 21).

[10] Vincenzo Maritati. *COS'È L'INGEGNERIA DELLE FUNZIONALITÀ*. URL: https://datamasters.it/blog/cose-lingegneria-delle-funzionalita/ (cit. a p. 37).

[11] dida. *What is Random Forest?* URL: `https://dida.do/what-is-random-forest` (cit. a p. 44).

[12] Erika Russi Eda Kavlakoglu. *Che cos'è XGBoost?* URL: `https://www.ibm.com/it-it/think/topics/xgboost` (cit. a p. 45).

[13] Data Science Team. *Cos'è la Light GBM?* URL: `https://datascience.eu/it/apprendimento-automatico/cose-la-light-gbm/` (cit. a p. 46).

[14] Okan Yenigün. *Smart Aspects of CatBoost Algorithm.* URL: `https://python.plainenglish.io/smart-aspects-of-catboost-algorithm-2720a6de4da6` (cit. a p. 47).

[15] Wikipedia. *Errore quadratico medio.* URL: `https://it.wikipedia.org/wiki/Errore_quadratico_medio` (cit. a p. 48).

Ringraziamenti

Se questa tesi fosse un viaggio, avrebbe avuto molte curve inaspettate, qualche tratto in salita e sicuramente alcune soste obbligate. Ma grazie a chi mi ha accompagnato lungo il percorso, sono finalmente arrivato al traguardo.

Un ringraziamento speciale va alla mia famiglia, alla quale devo tutto ciò che sono e che ho. Mi avete dato la possibilità di dimostrare quanto valgo, e spero di non avervi deluso in alcun modo. Riconosco ogni giorno i sacrifici che fate, mamma e papà, e per questo ho sempre vissuto con la "missione" di rendervi orgogliosi di me. Vedendovi oggi, credo di esserci riuscito almeno in parte. Spesso si attribuiscono tutti i meriti ai figli, ma la verità è che gran parte dei miei successi derivano dai vostri insegnamenti, dal vostro supporto e dai vostri sacrifici. Se sono qui, lo devo a voi.

Un pensiero speciale va a mia sorella. Questa volta sei qui con me, a condividere un momento che per me significa tanto. Ti auguro di vivere il tuo percorso con la stessa determinazione che dimostri ogni giorno. Ti prometto che sarò sempre il tuo primo tifoso, proprio come tu lo sei stata per me. Le aspettative ci sono, è vero, ma io non ho dubbi: farai grandi cose, e lo farai a modo tuo.

Un grande grazie ai miei nonni, che – nonostante la distanza generazionale e i cambiamenti tecnologici – si sono sempre interessati con affetto al mio percorso, anche se forse ancora oggi non hanno ben chiaro cosa abbia studiato o cosa farò in futuro.

Grazie anche ai miei zii, che mi hanno sempre seguito con affetto e che, soprattutto negli ultimi tempi, mi hanno offerto consigli preziosi su come affrontare il futuro.

Un ringraziamento particolare va a mio nonno, che non c'è più, ma che sono certo sarebbe orgoglioso di questo traguardo. Durante tutto il mio percorso accademico ho sempre sentito la responsabilità di onorarne la memoria, e sono felice di non averla disattesa.

Un grazie sincero alla mia compagnia di amici, con cui ho ricostruito un legame forte dopo un periodo di distacco. Grazie per le serate, le risate e, soprattutto, per essermi stati vicini nel momento più buio: il vostro supporto ha significato moltissimo per me.

Un ringraziamento speciale a Loris: insieme abbiamo condiviso esperienze indimenticabili, momenti di leggerezza ma anche confronti profondi. Sei come un fratello per me, uno di quei legami che non hanno bisogno di tante parole per essere capiti. In più di un'occasione sei stato un punto fermo, e sapere di poterti avere accanto è per me motivo di forza e serenità. Grazie per esserci sempre stato, con la tua presenza costante e autentica.

Un ringraziamento va anche a Francesca, che è entrata nella mia vita in un momento inaspettato, portando leggerezza, energia e nuovi stimoli. Con la tua presenza hai reso più semplice un periodo tanto intenso quanto importante. Ti ringrazio per i sorrisi, i confronti, la complicità e per avermi ricordato che, anche durante una corsa a ostacoli, c'è sempre spazio per respirare, guardarsi intorno e sentirsi bene.

Oltre alle persone che fanno parte della mia sfera più intima, voglio ringraziare anche chi ha condiviso con me il percorso accademico. In particolare, grazie ai miei compagni di corso, e ad Angelo, con cui ho stretto una vera amicizia e condiviso numerosi progetti durante la magistrale. Ho sempre ammirato la tua determinazione, la tua dedizione e la tua voglia di imparare: spesso ho cercato di prendere esempio da te per migliorarmi.

Un ringraziamento va anche ai ragazzi di Veicoli, con cui ho un legame ormai di lunga data e che mi hanno accolto nuovamente a braccia aperte, facendomi sentire a casa.

Una menzione speciale va a Enrico, che è diventato un fratello. Mi è stato vicino nel periodo più difficile dell'ultimo anno, è sempre stato una spalla su cui contare e una persona con cui confidarmi. Ha avuto a cuore me e il mio lavoro di tesi, e questo non lo dimenticherò mai.

Un grazie anche al mio relatore, che mi ha accolto come tesista nonostante il numero elevato di studenti. Non solo ha dimostrato interesse per il mio progetto, ma mi ha sempre guidato con disponibilità e gentilezza, aiutandomi a ottenere un risultato finale migliore.

Infine, voglio dedicare un pensiero a me stesso. Mi faccio i complimenti per il capitolo che ho chiuso. Ho sempre saputo di non essere un'eccellenza, ma ho riconosciuto in me determinazione e disciplina: qualità che mi hanno permesso di raggiungere i miei obiettivi. Negli ultimi due anni ho lavorato duramente su me stesso, sia in palestra che nello studio, cercando di dare sempre il massimo. Vedere oggi questo traguardo mi sembra quasi surreale: sapevo che ce l'avrei fatta, ma avevo anche paura di non riuscire o di non rispettare le scadenze.

Nonostante le lamentele degli ultimi cinque anni, so che conserverò questi ricordi con affetto e che un giorno li ripenserò con un sorriso. Alla fine, il mio percorso non è stato poi così tortuoso come quello di alcuni amici ed ex compagni.

Vorrei concludere a modo mio, con una frase tagliente e pungente:

"Everybody wants to know what I would do if I didn't win...
I guess we'll never know."

Impaziente di scoprire cosa mi riserverà il futuro.
Grazie di cuore a tutti, perché ognuno di voi ha contribuito a questo traguardo.

Matteo Bollo