Università Politecnica delle Marche

Ingegneria Informatica e dell'Automazione

Corso di

Manutenzione preventiva per la robotica e l'automazione intelligente

Rilevamento usura pala a partire dall'analisi del log di volo di un drone esarotore

Docente:
Alessandro Freddi

Studenti:
Caterina Sabatini
Matteo Stronati

Anno accademico 2025-2026

Indice

Elenco delle figure

1 Introduzione

La diffusione dei multirotori in applicazioni civili e militari, come ispezione, sorveglianza e trasporto, rende sempre più critico garantire alti livelli di sicurezza operativa. Secondo gli studi citati in "UAV-FD" [1], una quota significativa degli incidenti con un drone è riconducibile a problemi di equipaggiamento, in particolare a guasti degli attuatori e delle eliche, che possono portare rapidamente alla perdita di controllo del velivolo. In questo contesto, disporre di efficaci sistemi di *fault detection (FD)* diventa fondamentale per rilevare per tempo la presenza di danni e abilitare manovre di atterraggio sicuro, ma lo sviluppo e la validazione di tali algoritmi richiedono dataset pubblici, realistici e ben documentati, come il dataset utilizzato in questo progetto. Lo scopo è stato, quindi, impiegarlo come base sperimentale per addestrare e valutare modelli di classificazione binaria e successivamente ternaria per il rilevamento dei guasti, analizzandone anche il comportamento tramite tecniche di *explainability*.

1.1 Il dataset

Il dataset "UAV-FD" [1], raccoglie dati di volo reali di un esarotore soggetto a guasti di attuatori allo scopo di studiare il rilevamento di difetti alle eliche. Per rendere il dataset informativo rispetto alle vibrazioni generate dalle pale danneggiate, il firmware ArduPilot è stato modificato incrementando la frequenza di logging delle grandezze IMU da 25 Hz a circa 350 Hz, così da catturare sia la dinamica principale di rotazione (ordine delle decine di hertz) sia le armoniche associate allo sbilanciamento delle eliche. Le acquisizioni sperimentali comprendono 18 voli in ambiente esterno con condizioni di volo debole, eseguiti in modalità manuale da un operatore esperto. Ogni volo include decollo, tratto in volo e atterraggio. Sono state considerate tre condizioni operative: sei voli in assenza di guasti (*no-fault*), sei voli con un difetto al 5% su un singolo motore e sei voli con un difetto al 10% su un singolo motore. Il guasto viene realizzato tramite *chipping* artificiale delle pale accorciandole rispetttivamente del 5% e del 10% rispetto alla lunghezza nominale, ottenendo uno sbilanciamento realistico ma con impatto limitato sulla manovrabilità del velivolo.
I dati sono organizzati in 18 file `.mat`, uno per ciascun volo, i cui nomi indicano la condizione di fault e il motore interessato. Ogni file contiene i log di numerose grandezze: misure di accelerometri e giroscopi delle tre IMU, informazioni provenienti dagli ESC, comandi di controllo (uscite PWM, assetto desiderato e misurato), oltre alle stime fornite dai filtri EKF e alle variabili VIBE, che riassumono il livello di vibrazione percepito. Tutte le misure provengono da sensori già presenti a bordo, senza l'impiego di sensori aggiuntivi, il che rende il dataset rappresentativo di scenari realisticamente ottenibili su droni commerciali. Nel Capitolo 2 verranno descritte le operazioni di se-

lezione dei segnali e sincronizzazione temporale che portano dai log grezzi al dataset utilizzato per la generazione di feature prima e classificazione successivamente.

1.2 Obiettivi del progetto

Il progetto[1] si è posto i seguenti obiettivi, al fine di utilizzare in maniera completa il dataset fornito:

- **Classificazione binaria**: addestramento di modelli di apprendimento supervisionato in grado di distinguere tra condizioni *no-fault* e *fault*, aggregando in un'unica classe i guasti al 5% e al 10% dell'elica.
- **Classificazione ternaria**: estensione del problema a tre classi distinte (*no-fault*, *fault 5%*, *fault 10%*), valutando in particolare la capacità di discriminare tra diversi livelli di severità del guasto.
- **Analisi di explainability**: studio di come modelli con diversa complessità utilizzano le feature estratte dal dataset, tramite strumenti specifici, al fine di quantificare il trade-off tra accuratezza e interpretabilità.

[1] Il codice del progetto è disponibile su GitHub.

2 Preparazione e sincronizzazione dei dati

In questo capitolo viene descritta la fase di *preprocessing e sincronizzazione dei dati*, finalizzata alla loro preparazione per le successive fasi di analisi e classificazione dei fault. L'obiettivo è garantire la corretta sincronizzazione temporale dei segnali e l'uniformità dei processi di campionamento delle variabili impiegate nell'addestramento dei modelli di classificazione.

Tra le misure disponibili nel dataset sono state selezionate quelle ritenute maggiormente significative rispetto all'obiettivo prefissato. In particolare, sono state considerate le seguenti variabili:

- **IMU (Inertial Measurement Unit)**: sensore di accelerazioni e velocità angolari sui tre assi;
- **PWM (Pulse Width Modulation)**: segnali di comando ai motori proporzionali alla potenza richiesta;
- **ESC (Electronic Speed Controller)**: sensori che misurano velocità di rotazione e corrente dei motori;
- **ATTITUDE**: indica l'assetto del velivolo (rollio, beccheggio, imbardata);
- **XKF1 (Extended Kalman Filter)**: filtro che stima lo stato del drone;
- **VIBE (Vibration Sensors)**: sensori che rilevano vibrazioni strutturali e dei motori;
- **Time**: vettore temporale corrispondente alle misure di ciascun sensore.

2.1 Sincronizzazione dei dati

Prima della fase di sincronizzazione, viene impiegata la funzione `compareN` (Listato 2.1), che allinea i dati provenienti da più sensori sulla base di timestamp comuni, garantendo la coerenza temporale delle rispettive matrici dati.

```
function varargout = compareN(varargin)

N = nargin;
time = varargin{1}(:,2);

for k = 2:N
    time = intersect(time, varargin{k}(:,2), 'stable');
end
```

```
9
10 for k = 1:N
11     [~, ia] = ismember(time, varargin{k}(:,2));
12     varargout{k} = varargin{k}(ia,:);
13 end
14
15 end
```

Listato 2.1: *Codice della funzione* ***compareN***

La sincronizzazione dei dati è eseguita mediante la funzione `synchronize`, utilizzando l'opzione di unione dei timestamp (`union`) e la modalità *Zero-Order Hold* (`previous`). In questo modo, tutti i timestamp dei sensori vengono considerati, e i valori vengono mantenuti costanti fino al prossimo rilevamento disponibile, garantendo l'allineamento temporale delle misure senza introdurre distorsioni.

Successivamente, i dati sincronizzati vengono sottoposti a un resampling uniforme a 350 Hz tramite la funzione `retime`, con interpolazione lineare. Questo passaggio consente di ottenere un campionamento regolare necessario per le successive analisi e per l'addestramento dei modelli di classificazione dei fault. Una volta completata la sincronizzazione e il resampling, le variabili vengono organizzate per tipologia di sensore. Il codice utilizzato per implementare questa fase è riportato nel Listato 2.2.

```
1  %% SINCRONIZZAZIONE (ZOH)
2
3  all_tt_cell = {IMU_tt, PWM_tt, ATT_tt, XKF1_tt, VIBE_tt, ESC_tt_list
       {:}};
4  tt_sync = synchronize(all_tt_cell{:}, 'union', 'previous');
5
6  % Resample a 350 Hz uniforme
7  finishTime = seconds(tt_sync.Time(end));
8  timesout = seconds(0:1/Fs:finishTime)';
9  tt_resampled = retime(tt_sync, timesout, 'linear');
10
11 % Estrazione delle variabili
12 IMU_SYNC.GYR = [tt_resampled.IMU_GYR_X, tt_resampled.IMU_GYR_Y,
       tt_resampled.IMU_GYR_Z];
13 IMU_SYNC.ACC = [tt_resampled.IMU_ACC_X, tt_resampled.IMU_ACC_Y,
       tt_resampled.IMU_ACC_Z];
14 PWM_sync = tt_resampled.PWM;
15
16 for k = 1:num_motors
17     motor_id = k - 1;
18     ESC_SYNC.(sprintf('RPM%d', motor_id)) = tt_resampled.(sprintf('
           ESC_RPM%d', motor_id));
19     ESC_SYNC.(sprintf('CURR%d', motor_id)) = tt_resampled.(sprintf('
           ESC_CURR%d', motor_id));
20 end
21
22 ATTITUDE_SYNC.ROLL = tt_resampled.ATT_ROLL;
23 ATTITUDE_SYNC.PITCH = tt_resampled.ATT_PITCH;
```

```
ATTITUDE_SYNC.YAW = tt_resampled.ATT_YAW;
ATTITUDE_SYNC.DesROLL = tt_resampled.ATT_DesROLL;
ATTITUDE_SYNC.DesPITCH = tt_resampled.ATT_DesPITCH;
ATTITUDE_SYNC.DesYAW = tt_resampled.ATT_DesYAW;

XKF1_SYNC.ROLL = tt_resampled.XKF1_ROLL;
XKF1_SYNC.PITCH = tt_resampled.XKF1_PITCH;
XKF1_SYNC.YAW = tt_resampled.XKF1_YAW;
XKF1_SYNC.VN = tt_resampled.XKF1_VN;
XKF1_SYNC.VE = tt_resampled.XKF1_VE;
XKF1_SYNC.VD = tt_resampled.XKF1_VD;

VIBE_SYNC.ACC = [tt_resampled.VIBE_ACC_X, tt_resampled.VIBE_ACC_Y,
    tt_resampled.VIBE_ACC_Z];

time_sync = tt_resampled.Time;
```

Listato 2.2: *Codice della fase di sincronizzazione ZOH*

Successivamente, si procede con la rimozione delle fasi di decollo e atterraggio, che sono caratterizzate da comportamenti dinamici estremi e non rappresentativi del volo. Questa operazione è necessaria per garantire che i dati utilizzati per la classificazione siano privi di outlier dovuti a transitori iniziali o finali. Il processo di taglio dei dati avviene in tre fasi principali:

- *Throttle basso*: vengono identificati tutti i campioni in cui la somma dei segnali PWM dei motori è inferiore a una soglia prestabilita. Questi campioni corrispondono tipicamente al velivolo fermo o al minimo regime di motori e vengono eliminati da tutte le strutture sincronizzate.
- *Primi 2 secondi di volo*: i primi 2 secondi di dati vengono rimossi, in quanto rappresentano la fase iniziale del decollo, caratterizzata da transitori rapidi e instabilità dei segnali.
- *Ultimi 2 secondi di volo*: gli ultimi 2 secondi di dati vengono rimossi per eliminare la fase di atterraggio caratterizzata da condizioni dinamiche estreme che non sono rappresentative del volo stabile.

Infine, le variabili sincronizzate vengono organizzate all'interno della struttura `Data` e salvate in un file `.mat`, rendendo i dati pronti per le analisi successive.

2.2 Creazione della dataTable

Dopo la sincronizzazione dei dati, viene costruito un dataset per l'addestramento dei modelli di classificazione dei fault.

A ciascun file viene assegnato un codice di fault in base alla condizione del velivolo: assenza di fault (0) o presenza di fault (1) nel caso della classificazione binaria e assenza di fault (0%) o presenza di fault parziale (5%) o (10%) per la classificazione ternaria.

Per ogni file, il codice carica la struttura `Data` ed estrae tutte le variabili di interesse, quali *IMU*, *PWM*, *ESC*, *ATTITUDE*, *XKF1* e *VIBE*. Ciascuna variabile viene convertita in una *timetable* e inserita in una tabella MATLAB (`dataTable`), con una riga per file. Infine, a ogni riga viene associato il relativo codice di fault (`faultCode`), ottenendo un dataset completo e strutturato per l'addestramento dei modelli di classificazione.

3 Generazione delle feature

In questo capitolo viene descritto il processo di *feature engineering* adottato per trasformare i log grezzi dei voli, contenuti nella `dataTable` sincronizzata, in un insieme di variabili diagnostiche utilizzabili dai modelli di classificazione. A tale scopo è stato utilizzato il toolbox **Diagnostic Feature Designer** di **MATLAB** per estrarre sistematicamente feature sia nel dominio del tempo (statistiche di ampiezza e forma del segnale) sia nel dominio della frequenza (informazioni spettrali e di potenza). La procedura di generazione delle feature, descritta nelle sezioni successive, è stata mantenuta identica per entrambi i task di classificazione considerati nel progetto. In questo modo il confronto tra i modelli si basa sempre sullo stesso spazio delle feature, variando unicamente la definizione delle etichette di classe.

3.1 Importazione dei dati e frame policy

La generazione delle feature parte dalla `dataTable` sincronizzata ottenuta nel capitolo 2, che contiene per ciascun volo le misure dei sensori e il corrispondente codice di *fault*. Questa tabella viene importata nel toolbox Diagnostic Feature Designer selezionandola come sorgente dati principale; in fase di importazione è stato verificato che il campo `faultCode` venga riconosciuto correttamente come *condition variable*.

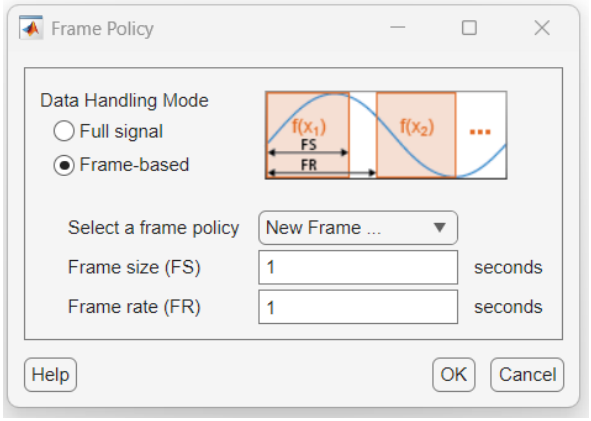


Figura 3.1: *Impostazione del pannello Frame Policy*

Nel pannello *Frame Policy* del toolbox è stata selezionata la modalità *Frame-based* e impostata una finestra di 1 s con frame rate pari a 1 s, come mostrato in Figura 3.1: in questo modo i segnali sono suddivisi in frame contigui, senza sovrapposizione, e per ciascun frame viene calcolato un vettore di feature. Questa scelta ha un duplice obiettivo: da un lato, aumentare il numero di realizzazioni disponibili per l'addestramento e il

test, poiché ogni finestra da 1 s viene trattata come un record indipendente; dall'altro, progettare e valutare l'algoritmo di *fault detection* su finestre temporali brevi, così da verificare se una diagnosi sufficientemente tempestiva possa essere ottenuta in vista di una possibile implementazione online.

3.2 Feature nel dominio del tempo

In una prima fase, il **Diagnostic Feature Designer** è stato utilizzato per estrarre feature nel dominio del tempo a partire dai principali segnali disponibili nella `dataTable` sincronizzata, comprendenti grandezze inerziali (accelerazioni e velocità angolari), comandi e misure degli attuatori (`PWM` e variabili degli `ESC`), variabili di assetto (`roll`, `pitch`, `yaw` e corrispondenti riferimenti) e indicatori di vibrazione forniti dal flight controller (`VIBE`). Per ciascun segnale e per ogni finestra definita dalla frame policy, sono state calcolate le classiche feature statistiche di segnale offerte dal toolbox, selezionando *media*, *valore RMS*, *deviazione standard*, *shape factor*, *kurtosis*, *skewness*, *crest factor*, *impulse factor*, *clearance factor* e *peak value*, come mostrato in Figura 3.2.

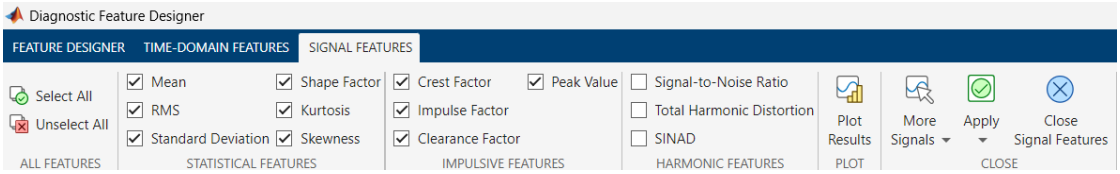


Figura 3.2: *Pannello di generazione delle feature nel tempo*

In questa fase non sono state considerate le feature di tipo armonico (ad esempio *Signal-to-Noise Ratio* e *Total Harmonic Distortion*), poiché l'analisi delle armoniche legate alla rotazione delle eliche viene affrontata in modo più mirato nella successiva Sezione dedicata alle feature nel dominio della frequenza. In questo modo le feature temporali descrivono principalmente l'andamento statistico e impulsivo dei segnali nelle finestre da 1 s, mentre le informazioni spettrali vengono concentrate e trattate separatamente nell'analisi in frequenza.

3.3 Feature nel dominio della frequenza

Per l'estrazione delle feature nel dominio della frequenza si è scelto di lavorare solo sui segnali effettivamente campionati ad alta frequenza dal flight controller, escludendo, dunque, le grandezze fornite dagli `ESC` e le correnti dei motori. Questi ultimi canali, infatti, nel dataset originale sono acquisiti a frequenze molto più basse rispetto alle grandezze inerziali; riportarli nella stessa griglia spettrale degli altri segnali avrebbe prodotto contenuti in banda alta dovuti unicamente alla procedura di sincronizzazione e non a reali componenti fisiche, introducendo così informazioni fuorvianti nelle feature in frequenza.

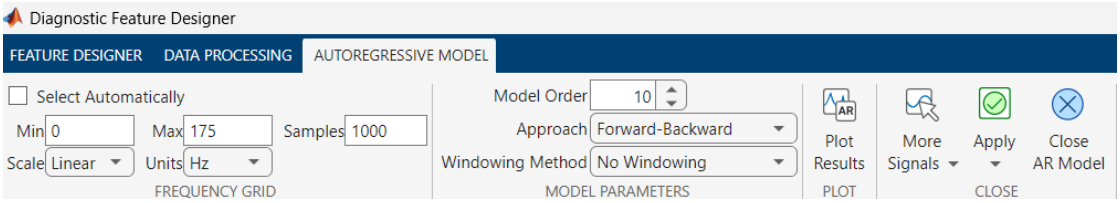


Figura 3.3: *Impostazione dell'Autoregressive Model*

Per i segnali selezionati, su ogni finestra di 1 s, è stata effettuata una stima spettrale mediante modello autoregressivo, utilizzando l'apposita sezione **Autoregressive Model** del toolbox. Come mostrato in Figura 3.3, il modello è stato configurato con ordine pari a 10, approccio Forward-Backward e nessuna finestratura addizionale. La scelta dell'ordine 10 riprende l'impostazione proposta nel lavoro originario sul dataset [1]. La griglia di frequenza è stata definita nell'intervallo 0–175 Hz, con 1000 campioni lineari, in modo coerente con il **teorema di Nyquist**, dato che la frequenza di campionamento dopo la modifica del firmware è pari a 350 Hz. In questo modo lo spettro AR stimato su ciascun frame cattura in maniera robusta le componenti principali legate alla rotazione delle eliche e alle armoniche associate a eventuali sbilanciamenti, fornendo la base per la successiva estrazione delle feature in frequenza.

Dopo aver stimato lo spettro tramite modello autoregressivo, le feature in frequenza sono state ottenute utilizzando il pannello **Spectral Features**. Come mostrato in Figura 3.4, per ogni segnale e per ciascun frame da 1 s è stata considerata una banda di frequenze compresa tra 15 Hz e 150 Hz, all'interno della quale il toolbox ricerca i picchi spettrali principali. Il limite inferiore a 15 Hz permette di escludere le componenti molto lente, mentre il limite superiore a 150 Hz è scelto per concentrarsi sulle frequenze associate alla rotazione dei motori e alle prime armoniche del moto delle eliche.

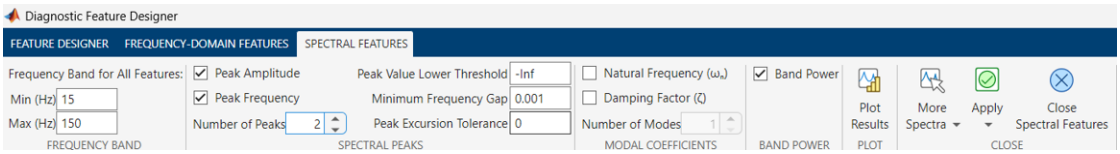


Figura 3.4: *Pannello di generazione delle feature in frequenza*

In questa banda, sono stati abilitati i parametri *Peak Amplitude* e *Peak Frequency*, con il numero di picchi impostato a 2, in modo da estrarre per ogni spettro le ampiezze e le frequenze delle due componenti dominanti. Questa scelta consente di catturare sia il tono principale legato alla velocità di rotazione dell'attuatore, sia una seconda armonica o eventuali risonanze significative, mantenendo al tempo stesso un numero contenuto di feature spettrali per segnale. Inoltre, è stata attivata la feature di *Band Power* sulla stessa banda 15–150 Hz, così da riassumere l'energia complessiva del contenuto vibratorio rilevante per la diagnosi del guasto.

3.4 Esportazione del workflow

Una volta definito l'insieme di feature nel dominio del tempo e della frequenza, il progetto del **Diagnostic Feature Designer** è stato esportato in codice MATLAB mediante l'opzione di generazione automatica della funzione di estrazione. In questo modo si è ottenuta la funzione `diagnosticFeatures.m` che implementa in forma programmatica tutte le operazioni configurate nell'interfaccia grafica: applicazione della frame policy da 1 s, calcolo delle feature temporali, stima spettrale autoregressiva e calcolo delle corrispondenti feature in frequenza.

Questa funzione viene richiamata sulla `dataTable` per produrre una tabella completa delle feature a livello di frame. Prima di procedere allo *split* in training e test è stato necessario effettuare una **fase di pulizia** per gestire i valori mancanti o non validi introdotti dal calcolo delle feature. In particolare, sono state mappate a "missing" tutte le occorrenze di valori infiniti e successivamente è stato conteggiato, per ciascuna feature, il numero di `NaN` presenti sulle righe. Le colonne con più del 25% di campioni mancanti sono state eliminate dalla tabella, mantenendo solo le feature sufficientemente popolate. Sulle feature rimanenti, il primo frame di ciascun volo è stato ripulito sostituendo eventuali `NaN` con il valore zero, mentre per tutti i frame successivi i valori mancanti sono stati riempiti con il valore numerico precedente lungo la stessa colonna. Successivamente sono state costruite le partizioni di training e test, mantenendo la stratificazione rispetto alle condizioni di *fault*. Si è ritenuto di utilizzare 70% del dataset per il training del modello e il restante 30% per la valutazione finale delle prestazioni.

L'intera pipeline di *feature engineering* è identica sia per il problema di classificazione binaria sia per quello ternario: ciò che cambia è esclusivamente la codifica della variabile di risposta (`faultCode`), aggregata in due classi nel primo caso e mantenuta in tre livelli nel secondo.

4 Classificazione

In questo capitolo viene illustrato il processo di classificazione. Come descritto nella Sezione 1.2, sono state eseguite due tipologie di classificazione: una binaria, finalizzata a distinguere tra le condizioni *no-fault* e *fault*, e una ternaria, volta a distinguere tra tre classi distinte: *no-fault*, *fault 5%* e *fault 10%*.

Per portare a termine gli obiettivi prefissati, è stato utilizzato lo strumento **Classification Learner** di **MATLAB** per addestrare e testare modelli di machine learning. Come descritto nella Sezione 3.4, per entrambe le classificazioni, il dataset è stato suddiviso tramite `holdout` in due partizioni, una per il training e uno per il testing dei modelli. In particolare, si è utilizzato il 70% del dataset per il training e il restante 30% per il testing, utilizzando il parametro `stratify` per garantire che le caratteristiche statistiche del dataset siano rappresentate equamente in entrambi gli insiemi. Per validare i modelli durante la fase di addestramento, è stato adottato il metodo di *K-fold Cross Validation*. Tale tecnica consiste nel suddividere il dataset di training in K sottoinsiemi: in ciascuna iterazione, un sottoinsieme viene utilizzato come set di validazione, mentre gli altri $K - 1$ costituiscono il set di addestramento. Questo procedimento garantisce che ogni sottoinsieme venga impiegato una volta come set di validazione. Nel caso specifico, all'interno del **Classification learner**, K è stato impostato con valore pari a 5.

Per eseguire la classificazione, è stata effettuata una selezione aggiuntiva delle *feature* tramite l'*Analysis Of Variance* (*ANOVA*). Questo metodo confronta la varianza delle singole *feature* tra le classi, consentendo di identificare e selezionare quelle che presentano la maggiore capacità discriminante. In Figura 4.1 è mostrato il risultato ottenuto mediante l'analisi. In particolare, per la classificazione è stato selezionato un numero di *feature* pari a 50.

Per ciascuna classificazione, sono stati addestrati tutti i modelli disponibili nel **Classification Learner**. Ai fini dell'analisi dei risultati, sono stati considerati i primi 10 modelli con la *accuracy* più elevata.

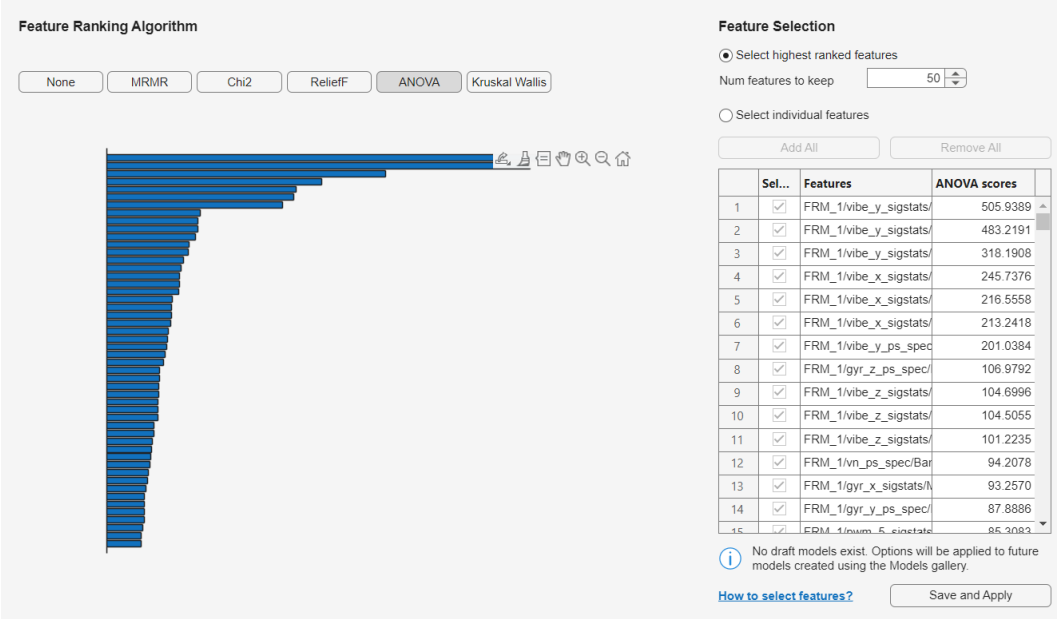


Figura 4.1: *Selezione delle feature con il metodo ANOVA*

4.1 Classificazione binaria

Prima di procedere con la classificazione binaria, è stato rilevato uno sbilanciamento nel dataset: i campi etichettati con `faultCode` pari a 1 risultano maggiori rispetto a quelli con valore pari a 0. Per compensare questa discrepanza, è stata impiegata la matrice dei *Misclassification Costs*, riportata in Figura 4.2. In particolare, è stato assegnato un costo maggiore, pari a 2.5, per l'errore commesso sulla classe minoritaria (0).

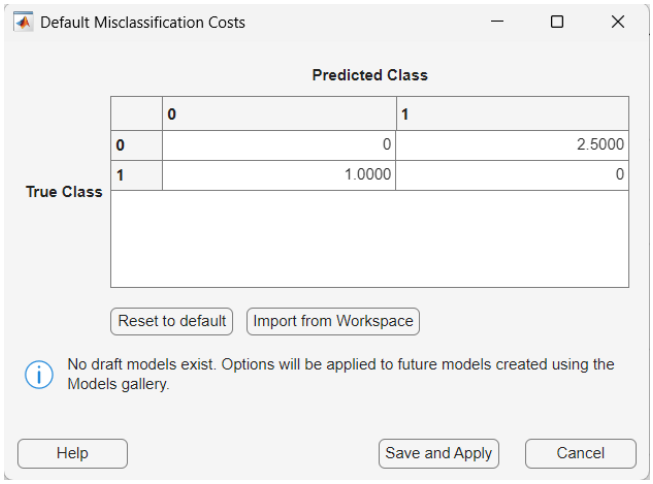


Figura 4.2: *Misclassification Costs*

La Figura 4.3 mostra i risultati della classificazione binaria relativi ai 10 modelli con la *accuracy* più elevata durante la fase di test. Il modello che ha ottenuto le migliori prestazioni è il **Boosted Trees**, con un punteggio di *accuracy* pari al 96.9%. Tuttavia, si osserva che la performance di questo modello non differisce in maniera significativa rispetto ai modelli immediatamente successivi.

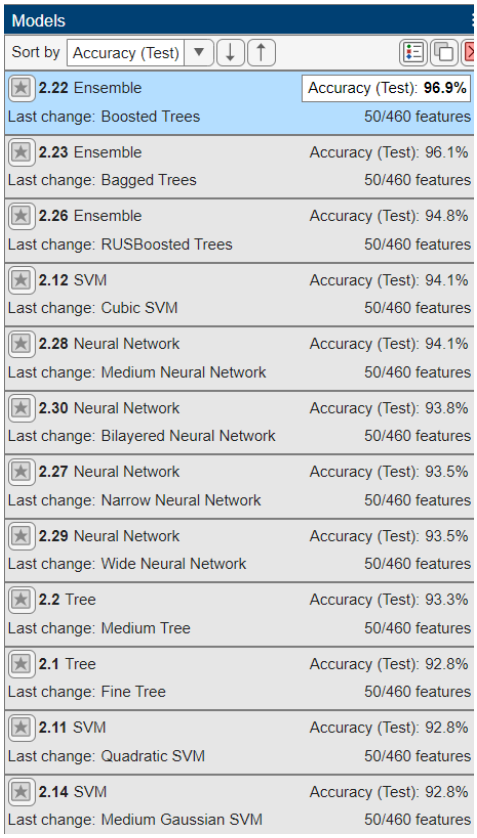


Figura 4.3: *Risultati della classificazione binaria*

Come mostrato in Figura 4.4 nella matrice di confusione, il modello riconosce correttamente il 95% delle istanze della classe 0 e il 95,7% delle istanze della classe 1. Il tasso di errore risulta pertanto molto basso per entrambe le classi, attestandosi intorno al 4-5%.

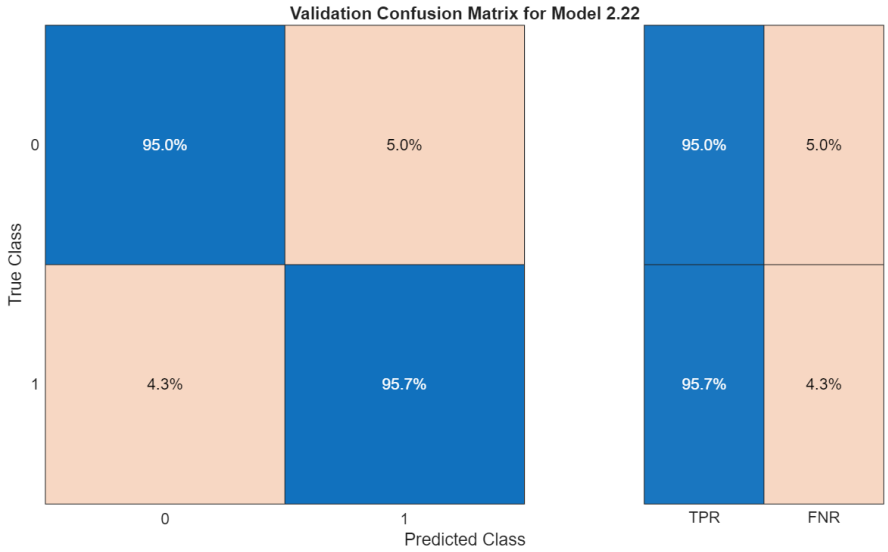


Figura 4.4: *Matrice di confusione del Boosted Trees*

4.2 Classificazione ternaria

Nel caso della classificazione ternaria, i risultati sono riportati in Figura 4.5. In particolare, sono riportati i primi 10 modelli con valore di *accuracy* più elevata.

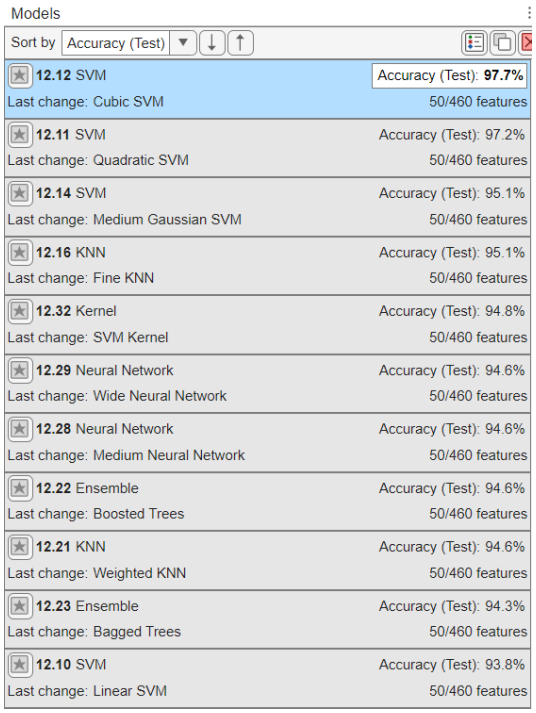


Figura 4.5: *Risultati della classificazione ternaria*

Il modello che ha ottenuto le migliori prestazioni in questo caso è il **Cubic Support Vector Machine** (**SVM Cubic**), con un punteggio di *accuracy* pari al 97.7%. Analogamente a quanto osservato nella classificazione binaria, anche gli altri modelli presentano valori di *accuracy* molto simili, indicando che le prestazioni complessive dei modelli più accurati sono sostanzialmente comparabili. Come indicato in Figura 4.6, nella quale è mostrata la matrice di confusione relativa al SVM Cubic, il modello raggiunge un 98.2% per la classe 0, 95.9% per la classe 5 e 98.7% per la classe 10. Anche gli errori compiuti sono minimi con una percentuale che oscilla tra 1% e il 4%.

I risultati ottenuti mostrano una buona coerenza con quanto riportato negli studi del "UAV-FD" [1]. Infatti, in entrambi i casi, le prestazioni dei modelli di classificazione risultano elevate.

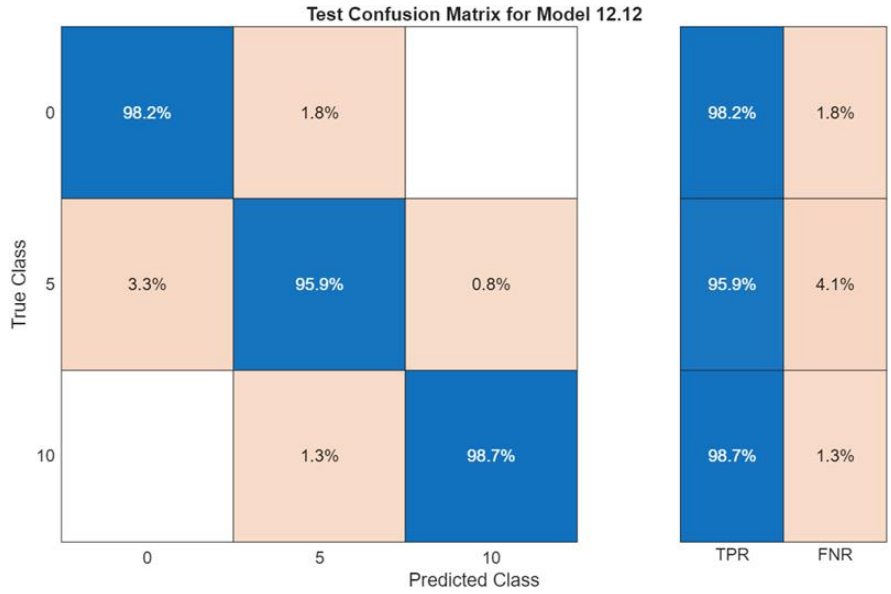


Figura 4.6: *Matrice di confusione del SVM Cubic*

Per approfondire l'analisi effettuata è stato calcolato il *Fault Isolation Rate* (*FIR*), il quale rappresenta la percentuale di guasti che il classificatore è in grado di isolare in modo non ambiguo. Esso viene calcolato come:

$$\text{FIR} = \frac{A_t}{A_t + C_t} \cdot 100$$

dove:

- $A_t = \sum_i A_i$ rappresenta il numero di guasti che il sistema è in grado di isolare in modo corretto;
- $C_t = \sum_i C_i$ indica il numero di guasti che il sistema non è in grado di isolare correttamente.

Nel caso del SVM Cubic, il FIR ha valore pari a 98.9%, indicando la capacità di classificazione ottimale dell'algoritmo.

4.2.1 Classificazione ternaria mediante PCA

La classificazione ternaria è stata ripetuta utilizzando la *Principal Component Analysis* (*PCA*), un metodo che permette di ridurre il numero di *feature* proiettandole su un insieme più ristretto di variabili, dette *Componenti Principali*, caratterizzate dalla massima varianza. L'obiettivo di questa trasformazione è preservare quanta più informazione possibile con un numero ridotto di dimensioni, semplificando il modello e migliorandone l'efficienza computazionale.

In Figura 4.7 è riportata l'impostazione dei parametri adottati per la *PCA*. In particolare, il parametro *Explained variance (percent)* è stato fissato al 99%, indicando al software di selezionare il numero minimo di componenti principali necessario a spiegare il 99% della variabilità totale dei dati.

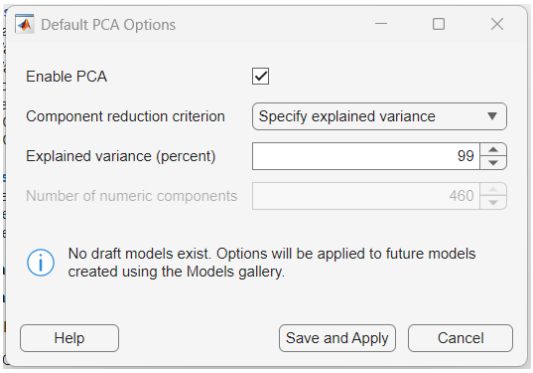


Figura 4.7: *Impostazione della PCA*

I risultati ottenuti risultano coerenti con quanto riportato nello studio "UAV-FD" [1]. In particolare, l'utilizzo della *PCA* tende a determinare una riduzione dell'*accuracy* dei modelli, poiché la proiezione nello spazio delle componenti principali comporta una perdita parziale di informazione. Di conseguenza, lo spazio delle *feature* risulta meno informativo rispetto a quello originale.

Le prestazioni migliori sono ottenute mediante classificatori non lineari, in quanto maggiormente in grado di catturare relazioni complesse tra le variabili. Tuttavia, anche il modello più performante, il *Wide Neural Network*, raggiunge un valore massimo di *accuracy* pari all'80.4%, evidenziando una riduzione significativa delle prestazioni rispetto ai modelli addestrati nelle analisi precedenti.

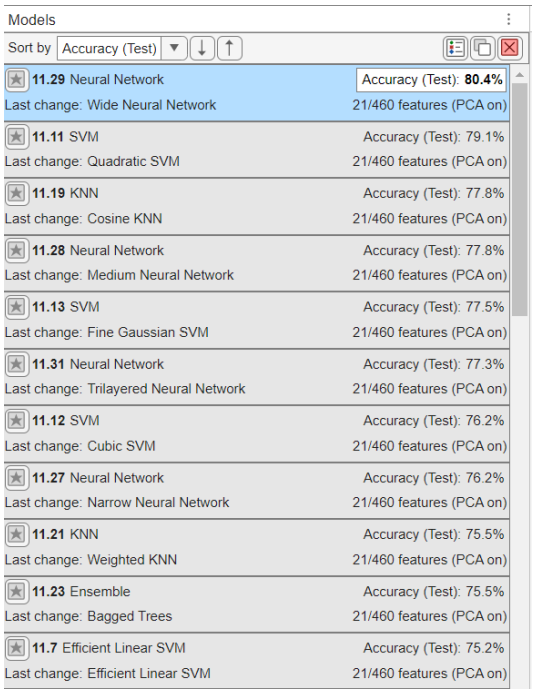


Figura 4.8: *Risultati della classificazione con PCA*

La matrice di confusione relativa al *Wide Neural Network*, riportata in Figura 4.9, evidenzia prestazioni significativamente inferiori rispetto ai risultati ottenuti senza l'impiego della *PCA*. In particolare, si osserva un marcato incremento del tasso di errore, che risulta compreso tra l'11% e il 28%, indicando una ridotta capacità del modello di discriminare correttamente le diverse classi nello spazio delle feature ridotto.

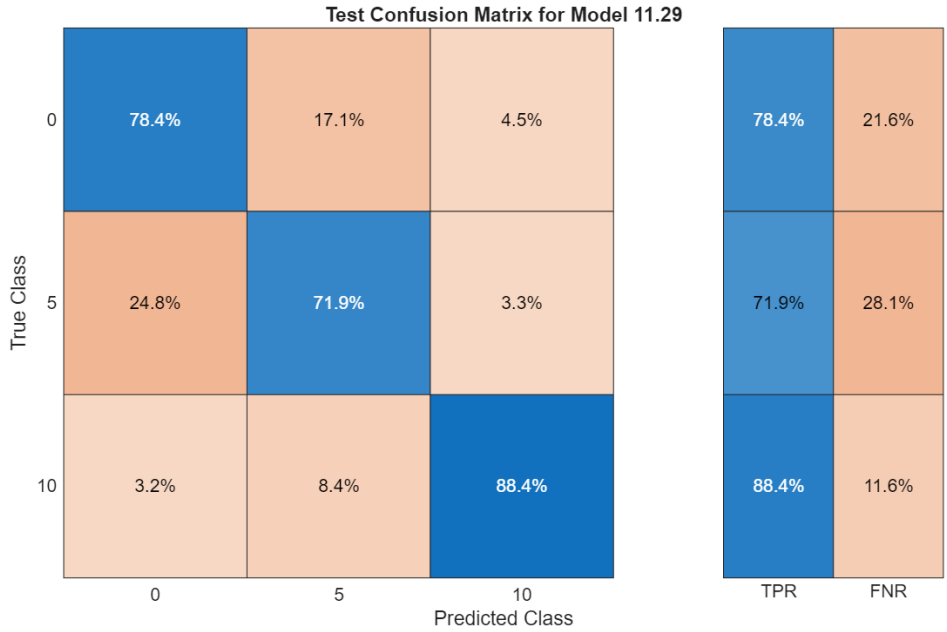


Figura 4.9: *Matrice di confusione della Wide Neural Network*

5 Analisi dell'explainability

L'explainability riveste un ruolo cruciale nella diagnosi di guasti industriali, specialmente in applicazioni safety-critical. Come dimostrato in una review sistematica [2], i modelli di machine learning "black-box", come SVM e reti neurali, presentano limitazioni significative in termini di *trust* degli operatori e conformità normativa (*EU AI Act*), poiché non forniscono spiegazioni comprensibili delle loro decisioni. Per questo motivo, tecniche di XAI (Explainable AI) come Shapley values, LIME e visualizzazioni interpretative sono diventate standard per rendere trasparenti le predizioni, identificando quali pattern nei dati guidano la classificazione dei *fault*.
Anche nel presente progetto si è reso, dunque, necessario un approccio sistematico all'explainability, motivato dalla necessità di validare fisicamente i risultati per comprendere il motivo fisico dietro ogni allarme fault generato per interventi mirati di manutenzione predittiva. L'analisi di explainability è stata condotta esclusivamente sul caso di classificazione ternaria, poiché rappresenta lo scenario operativo più realistico e critico.

Tutte le analisi di explainability sono state condotte utilizzando la sezione **Explain** del **Classification Learner** toolbox di **MATLAB**, che integra tecniche standard come *permutation importance*, *partial dependence plots* e *Shapley summary plots*.

Model Number	Model Type	Status	Accuracy (Test)	Prediction Speed (obs/sec)	Compact Model Size (bytes)	Coder Model Size (bytes) ↑
12.3	Tree	Tested	83.98 %	17191	13915	9950
12.2	Tree	Tested	90.70 %	9078.5	18599	11930
12.7	Efficient Linear SVM	Tested	72.61 %	10872	71066	12822
12.6	Efficient Logistic Regression	Tested	92.25 %	10328	71216	12846
12.1	Tree	Tested	91.47 %	8806.8	21595	13118
12.10	SVM	Tested	93.80 %	11716	54401	15300
12.27	Neural Network	Tested	93.54 %	13116	19612	15350
12.30	Neural Network	Tested	92.76 %	15348	21384	16598
12.31	Neural Network	Tested	93.02 %	16732	23156	17846
12.28	Neural Network	Tested	94.57 %	14525	26092	21830
12.4	Discriminant	Tested	92.76 %	14599	57824	33928
12.8	Naive Bayes	Tested	81.65 %	9531.8	39972	35919
12.29	Neural Network	Tested	94.57 %	14183	58492	54230
12.5	Discriminant	Tested	86.56 %	17036	147106	77150
12.22	Ensemble	Tested	94.57 %	4156.1	573444	102041
12.26	Ensemble	Tested	93.28 %	4698.5	573444	102041
12.11	SVM	Tested	97.16 %	11331	166793	127740
12.12	SVM	Tested	97.67 %	13912	184265	145212

Figura 5.1: *Tabella riassuntiva dei modelli di classificazione ternaria*

La tabella, raffigurata in Figura 5.1, mostra i modelli di classificazione addestrati, ordinati per dimensione crescente del modello compilato, criterio fondamentale per valutare la fattibilità pratica di *deploy* su hardware embedded. Come si può notare,

sono stati scelti due modelli per fare un confronto di metriche ed explainability. Le motivazioni di questa scelta si possono riassumere così:

- **SVM cubic**: È il modello che emerge per la massima accuracy in test (97.67%), ma presenta il limite tipico dei modelli black-box di non essere interpretabile nativamente e una dimensione in bytes piuttosto grande.
- **Fine Tree**: È modello molto leggero ideale per edge computing, nonostante offra un'accuracy minore rispetto all'altro (91.47%). Ha, però, il grande vantaggio intrinseco di *explainable* grazie a regole *if-then* immediatamente comprensibili, come affermato in letteratura [3].

Questa combinazione permette di confrontare un modello black-box ottimale (SVM) con un modello intrinsecamente interpretabile (Fine Tree), validando l'explainability su due estremi, per esaminare il tema nel modo più completo possibile.

5.1 Metodologia implementata

Per valutare l'explainability dei due modelli selezionati, sono stati condotti confronti diretti tra SVM cubic e Fine Tree attraverso cinque tecniche complementari:

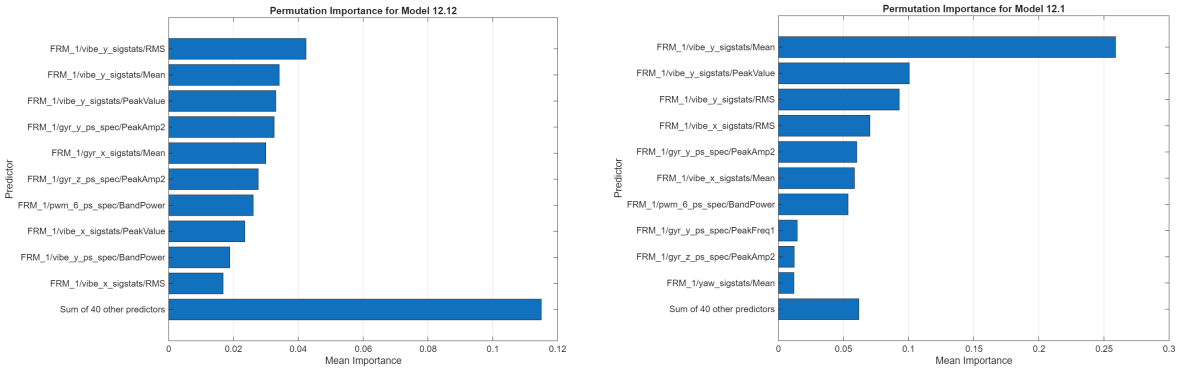
- Permutation Importance;
- Shapley Importance caratterizzata per le classi;
- Shapley Summary caratterizzata per le classi;
- Partial Dependence Plots su due feature importanti;
- LIME applicato a due istanze esemplificative (una predizione corretta e una errata).

Nelle sezioni successive vengono presentati tutti i grafici di confronto tra i due modelli accompagnati da un'analisi dettagliata dei risultati, con particolare attenzione alla coerenza delle spiegazioni fisiche emerse.

5.2 Permutation Importance

La Permutation Importance quantifica l'importanza globale di ciascuna feature misurando la degradazione dell'*accuracy* quando i suoi valori vengono permutati casualmente nel test set: feature critiche provocano cali significativi di performance, emergendo così come discriminanti principali per la classificazione dei *fault*. Questa tecnica rivela quali caratteristiche guidano sistematicamente le predizioni, indipendentemente dall'algoritmo sottostante, e permette il confronto diretto tra Fine Tree e SVM Cubic

per validare la robustezza delle spiegazioni fisiche. A tal proposito, in Figura 5.2, è proposto il confronto dei grafici di Permutation Importance tra i due modelli.



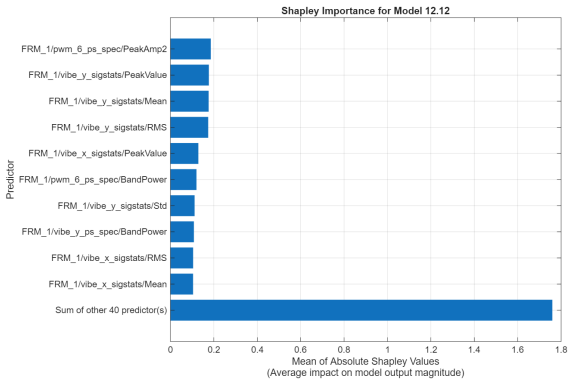
(a) *Permutation Importance - SVM Cubic* **(b)** *Permutation Importance - Fine Tree*

Figura 5.2: *Confronto dei grafici di Permutation Importance*

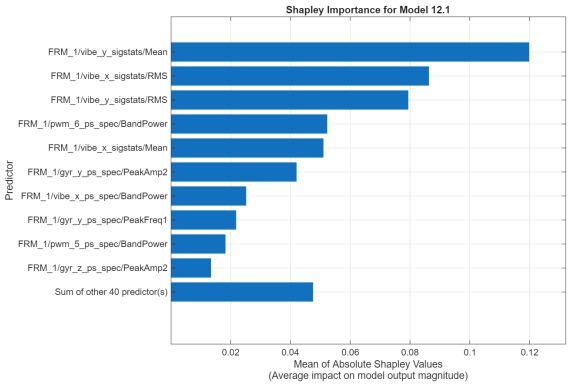
Il confronto diretto della Permutation Importance tra SVM Cubic e Fine Tree evidenzia convergenze fisiche e differenze algoritmiche che confermano la **superiorità interpretativa del Fine Tree**. Entrambi identificano i dati di vibrazione come discriminante primaria, coerente con la fisica dell'usura pala. L'SVM Cubic distribuisce le importanze su molte feature con una scala uniforme, mentre il Fine Tree concentra drasticamente l'importanza su tre-quattro variabili dominanti con crollo netto verso 0. Questa concentrazione naturale, intrinseca alla struttura ad albero, rende il Fine Tree più stabile e leggibile rispetto alla dipendenza multi-feature dell'SVM.

5.3 Shapley Importance

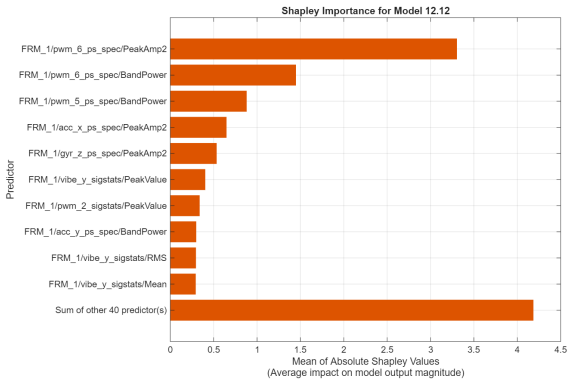
La Shapley Importance quantifica i contributi medi di ciascuna feature alle predizioni del modello: per ogni feature si calcola quanto marginalmente contribuisce alla spiegazione della classe predetta, producendo un ranking per singola classe. Questa peculiarità consente di caratterizzare come le feature discriminano specificamente ogni stato operativo: ad esempio, quali misure "spingono" verso *no-fault* e quali indicano *fault severo*. In Figura 5.3 è proposto il confronto dei grafici di Shapley Importance tra i due modelli per ciascuna classe.



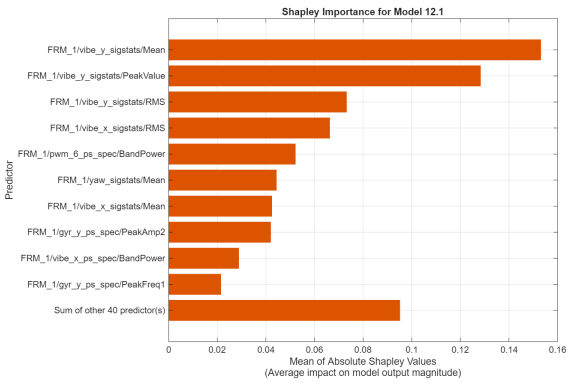
(a) *Shapley Importance - SVM Cubic classe 0*



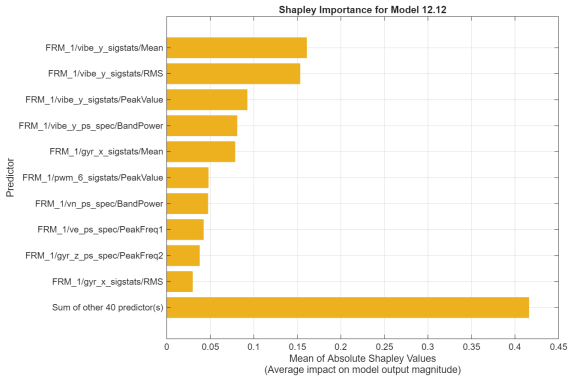
(b) *Shapley Importance - Fine Tree classe 0*



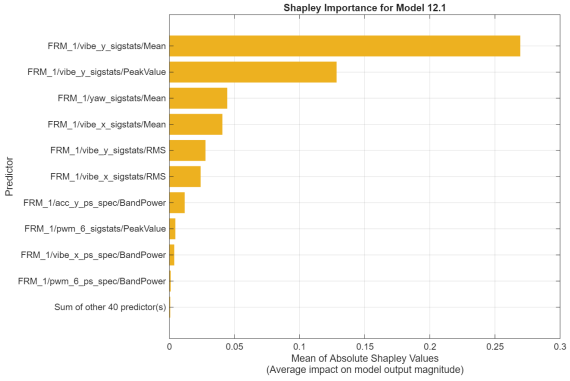
(c) *Shapley Importance - SVM Cubic classe 5*



(d) *Shapley Importance - Fine Tree classe 5*



(e) *Shapley Importance - SVM Cubic classe 10*



(f) *Shapley Importance - Fine Tree classe 10*

Figura 5.3: *Confronto dei grafici di Shapley Importance*

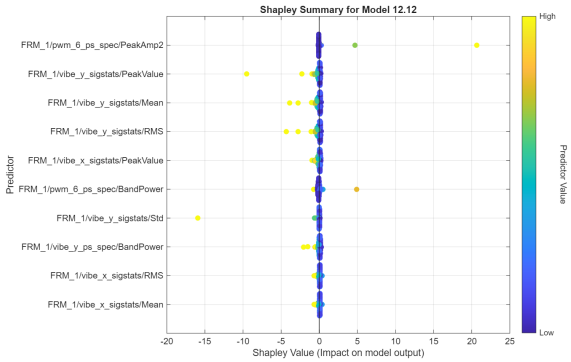
I grafici di Shapley Importance per ciascuna classe confermano la robustezza fisica delle spiegazioni attraverso convergenza e differenze algoritmiche. Una differenza chiave emerge nelle feature per le classi 0 e 5: nell'SVM Cubic sono visibili e rilevanti contributi da feature derivate `PWM` (es. `PeakAmp`), mentre il Fine Tree resta sempre dominato dalle vibrazioni pure, subendo meno influenza da altre misure. Si nota che le feature

derivate dai valori di **vibrazione sull'asse y** sono, in generale, piuttosto dominanti sia per la Permutation Importance che per la Shapley Importance (in tutte le classi); ciò è coerente con la progressione vibrazionale sull'asse y data dall'usura della pala. Osservando i grafici, emerge ancora una volta la maggiore spiegabilità del Fine Tree: la scala delle sue barre è marcatamente più polarizzata, con poche feature che catturano la quasi totalità della discriminazione, contro la distribuzione più uniforme e sfumata dell'SVM. Tale concentrazione netta rende immediatamente comprensibili le logiche decisionali dell'albero, confermando la sua superiorità interpretativa.

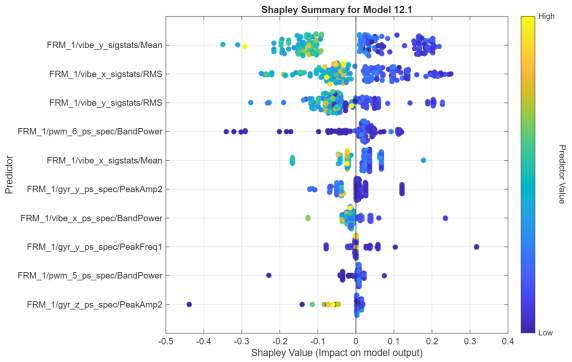
5.4 Shapley Summary

La Shapley Summary visualizza la distribuzione dei *valori Shapley* per ciascuna feature sul dataset, mostrando come i valori delle feature influenzino le predizioni del modello attraverso impatti positivi o negativi. Ogni punto rappresenta il contributo di una feature per un'istanza specifica, con l'asse x che indica l'impatto sul risultato: valori positivi spingono verso la classe predetta, negativi la allontanano. Per la visualizzazione sono state selezionate le 10 feature con Shapley Importance maggiore. I colori indicano il valore della feature: blu per valori bassi, giallo per alti seguendo la scala rappresentata. Per capire l'effetto, vanno combinati posizione e colore: punti gialli a sinistra significano che alti valori della feature riducono la probabilità della classe; blu a destra indicano che bassi valori la aumentano. In Figura si può apprezzare il confronto dei grafici di Shapley Summary tra i due modelli per ciascuna classe.

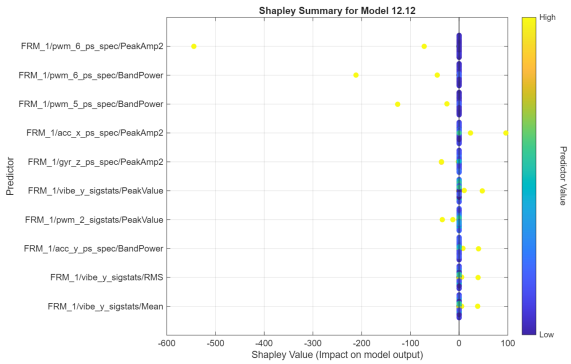
Si nota che i grafici per SVM Cubic sulle classi 0 e 5 non sono valutabili per via degli outlier estremi che dominano l'asse x, rendendo invisibili i pattern centrali: questo è già un sintomo di bassa explainability e interpretabilità del modello, che dipende da pochi casi anomali estremi invece di trend generali. Al contrario, Fine Tree mostra distribuzioni bilanciate e compatte per tutte le classi, permettendo di individuare pattern coerenti. Ad esempio, confrontando i tre grafici per l'albero decisionale, si nota che valori bassi della feature `vibe_y_Mean` caratterizzano le classi 0 e 5, mentre valori alti identificano nettamente la classe 10. Questa è una riflessione molto importante che conferma l'idea di alta interpretabilità del modello Fine Tree. In più, per la classe 10 entrambi i modelli mostrano distribuzioni simili ma l'albero mostra pattern più visibili e marcati, al contrario di SVM Cubic che presenta una sfumatura a tutti gli effetti.



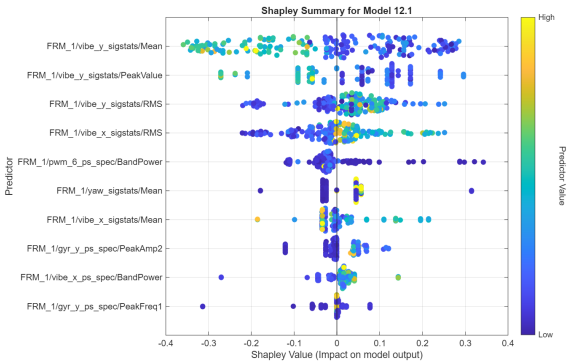
(a) *Shapley Summary - SVM Cubic classe 0*



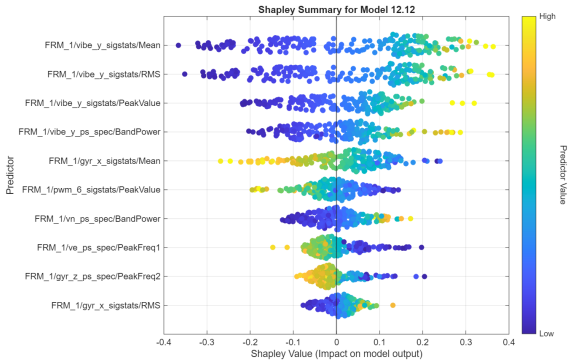
(b) *Shapley Summary - Fine Tree classe 0*



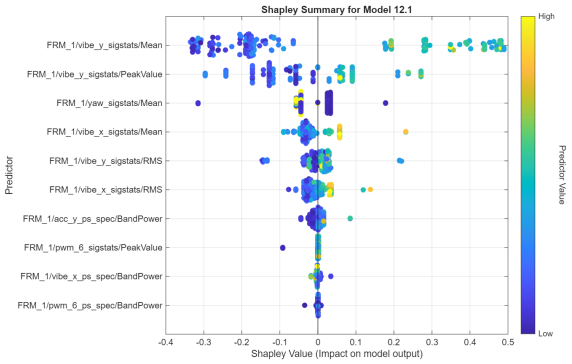
(c) *Shapley Summary - SVM Cubic classe 5*



(d) *Shapley Summary - Fine Tree classe 5*



(e) *Shapley Summary - SVM Cubic classe 10*



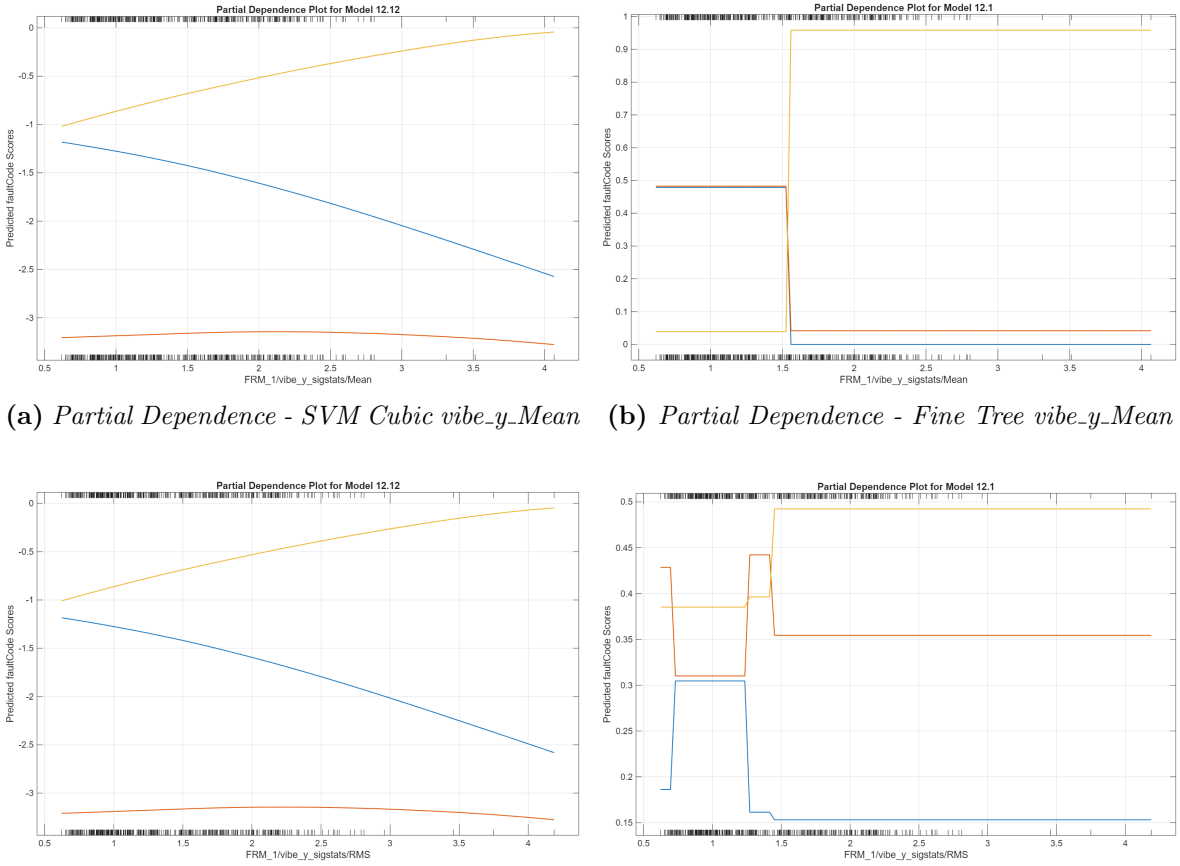
(f) *Shapley Summary - Fine Tree classe 10*

Figura 5.4: *Confronto dei grafici di Shapley Summary*

5.5 Partial Dependence

La Partial Dependence visualizza l'effetto marginale medio di una feature specifica, mediando sugli effetti delle altre feature. Si fissa la feature sull'asse x e si calcola la predizione media variando solo quella, tenendo le altre fisse ai loro valori reali tramite

il metodo *ICE (Individual Conditional Expectation)*, poi aggregato in linee medie per classe. Nel grafico ogni colore rappresenta una classe diversa: blu per Classe 0, arancione per Classe 5 e giallo per classe 10. L'asse y indica lo score di predizione: valori alti positivi significano alta probabilità per la classe, negativi indicano bassa probabilità. Le "lineette nere" sull'asse x sono una rappresetnazione della distribuzione empirica dei dati per quella feature, mostrando la densità dei punti osservati. Nella Figura 5.5 sono presenti i grafici, relativi ai due modelli, di Partial Dependence per due feature con alto valore di Permutation Importance: `vibe_y_Mean` e `vibe_y_RMS`.



(a) *Partial Dependence - SVM Cubic vibe_y_Mean* **(b)** *Partial Dependence - Fine Tree vibe_y_Mean*

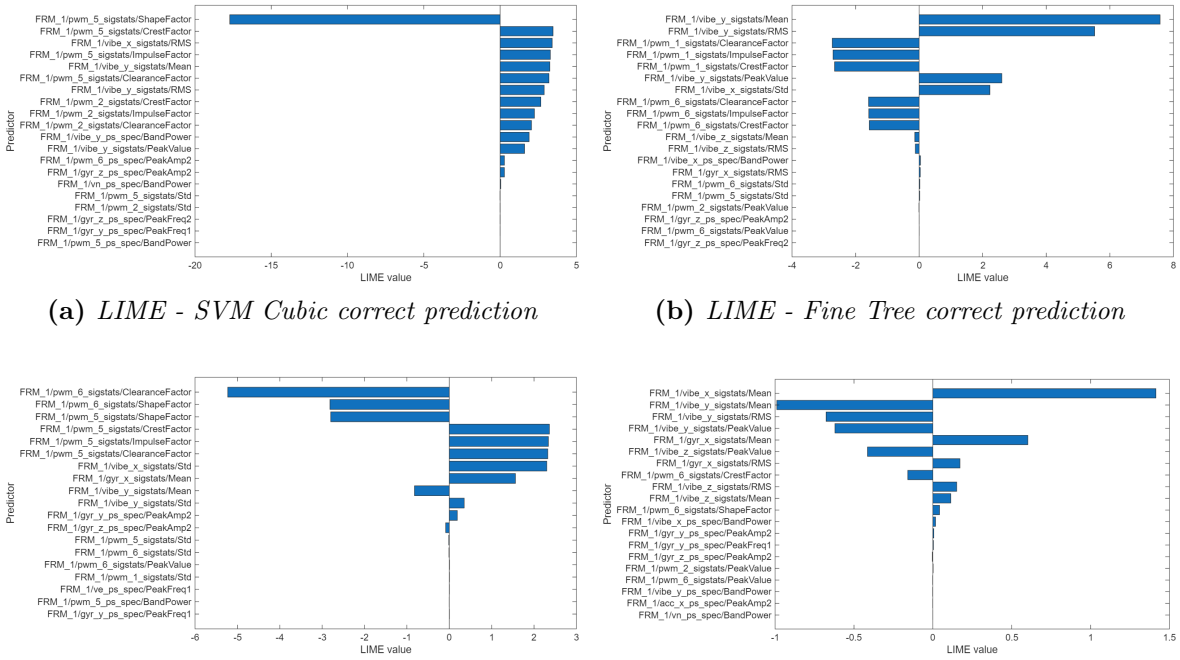
(c) *Partial Dependence - SVM Cubic vibe_y_RMS* **(d)** *Partial Dependence - Fine Tree vibe_y_RMS*

Figura 5.5: *Confronto dei grafici di Partial Dependence*

I grafici di confronto delle Partial Dependence tra i due modelli suggeriscono riflessioni interessanti: Fine Tree produce curve con divisioni nette a partire da valori bassi della feature, mostrando trend chiari che riflettono regole decisionali semplici, aumentando l'interpretabilità del modello. Molto interessante risulta essere il grafico della Partial Dependence del Fine Tree per la feature `vibe_y_Mean` che, per la classe 10, ha una curva a gradino quasi perfetto, indicando come una soglia di valore poco superiore a 1.5 oltre la quale aumenta drasticamente la predizione della classe 10, indicando che questa è una feature davvero importante per il modello.

5.6 LIME

LIME (Local Interpretable Model-agnostic Explanations) è una tecnica di explainability locale che spiega singole predizioni dei modelli, approssimando localmente il loro comportamento con un surrogato semplice e interpretabile, tipicamente una regressione lineare pesata. Per funzionare, LIME seleziona un'istanza specifica, genera perturbazioni locali attorno a quel punto modificando i valori delle feature con rumore gaussiano o uniforme, calcola le predizioni del modello originale su queste perturbazioni vicine e addestra un modello lineare solo nel loro intorno locale. Il valore assegnato a ogni feature è precisamente il coefficiente del surrogato lineare, che quantifica il contributo marginale di quella feature alla predizione locale: positivi aumentano la probabilità della classe predetta, negativi la riducono, con la magnitudine che indica l'importanza relativa.
L'analisi LIME è stata condotta per fornire spiegazioni locali complementari alle altre tecniche e confrontare l'explainability tra i modelli, analizzando per entrambi: un'istanza correttamente predetta della classe 10 e un'istanza con *true value* 0 ma predizione errata classe 5.



(a) *LIME - SVM Cubic correct prediction*

(b) *LIME - Fine Tree correct prediction*

(c) *LIME - SVM Cubic wrong prediction*

(d) *LIME - Fine Tree wrong prediction*

Figura 5.6: *Confronto dei grafici di LIME*

In Figura 5.6 sono rappresentati i grafici LIME per i due modelli nelle due istanze descritte in precedenza. In alto si possono osservare i grafici relativi a una predizione corretta della classe 10. Si nota che il modello SVM Cubic ha successo attraverso un meccanismo di bilanciamento tra forze opposte: il `pwm_5_sigstats/ShapeFactor` eser-

cita un contributo negativo estremo, che viene compensato da un insieme distribuito di feature positive. Questo schema è fisicamente ambiguo: lo ShapeFactor del motore 5 sembra tipico di un'altra classe, e il modello lo "corregge" aggregando molti altri segnali. Dal punto di vista dell'explainability, questo rende la predizione meno trasparente e potenzialmente fragile in generalizzazione. Il Fine Tree, al contrario, basa la decisione su una logica semplice e fisicamente coerente: `vibe_y_sigstats/Mean` e `vibe_y_sigstats/RMS` dominano nettamente. I contributi negativi sono presenti ma contenuti e non alterano la logica complessiva. La regola appresa è fisicamente sensata: un guasto dell'elica al 10% si manifesta con energia vibrazionale elevata sull'asse y, esattamente ciò che ci si aspetta da uno squilibrio meccanico incipiente.

In basso si osservano i grafici relativi a una predizione errata con *true value* di 0 ma classificato con la classe 5. L'SVM Cubic arriva alla predizione sbagliata attraverso un meccanismo già visto: compensazione tra contributi opposti, ma questa volta sbilanciata nel verso errato. Questo è un caso classico di predizione fragile: il modello "vince" per la somma di tanti piccoli contributi positivi che sovrastano pochi ma forti segnali negativi. Fisicamente non c'è una motivazione coerente, il modello si è agganciato a pattern del segnale PWM del motore 5 che nel dato nominale assomigliano accidentalmente a quelli di un guasto al 5%. Il Fine Tree commette lo stesso errore ma con una logica più trasparente: il modello è ingannato da valori di vibrazione e giroscopio sull'asse x, probabilmente per un disturbo o una manovra del drone. Tuttavia, le feature che solitamente discriminano maggiormente la presenza del guasto (vibrazione sull'asse y) spingono verso la predizione della classe corretta 0, avendo contributi negativi nel valore di LIME. In conclusione, appare che il Fine Tree sbaglia in modo comprensibile e spiegabile, mentre l'SVM Cubic sbaglia in modo "opaco" a causa di un equilibrio instabile tra tanti contributi PWM.

6 Conclusioni

Il progetto ha affrontato il problema della diagnosi di guasto in un drone esarotore mediante l'analisi del dataset "UAV-FD" [1], con l'obiettivo di sviluppare e validare modelli di classificazione supervisionata capaci di operare su due livelli distinti. In una prima fase, i modelli sono stati progettati per distinguere tra condizioni di funzionamento nominali e condizioni anomale, così da individuare la presenza di un guasto. In una seconda fase, una volta rilevata l'anomalia, il sistema è stato addestrato a riconoscere e differenziare le diverse tipologie di degrado della pala.

In particolare, dopo una fase di *preprocessing* del dataset, durante la quale i dati sono stati sincronizzati e sottoposti a operazioni di pulizia per garantirne coerenza e qualità, si è proceduto all'estrazione delle feature tramite il **Diagnostic Feature Designer**. Le caratteristiche ottenute, calcolate sia nel dominio del tempo sia nel dominio della frequenza, sono state successivamente utilizzate come riferimento per i modelli di classificazione. Successivamente, attraverso lo strumento **Classification Learner** sono stati estratti i risultati di interesse.

Pur con prestazioni leggermente variabili tra i modelli, i risultati ottenuti in tutte le fasi di classificazione si sono rivelati complessivamente ottimali. Sia nella classificazione binaria sia in quella ternaria, l'*accuracy* dei modelli ha superato il 90%. In particolare, per la classificazione binaria il modello con le migliori prestazioni è risultato il **Boosted Trees**, mentre per quella ternaria si è distinto il **SVM Cubic**. Inoltre, per la classificazione ternaria è stato condotto un esperimento aggiuntivo utilizzando la *PCA*, che tuttavia ha portato a risultati inferiori rispetto all'approccio originale.

Infine, è stata analizzata l'*explainability* di alcuni dei modelli più significativi sviluppati per la classificazione ternaria, al fine di comprendere come venissero generate le singole predizioni e quali caratteristiche dei dati avessero il maggiore impatto sulla distinzione tra le classi. In particolare, sono stati analizzati due algoritmi: il **SVM Cubic**, già citato in precedenza e scelto per l'elevata *accuracy* nelle predizioni, e il **Fine Tree**, selezionato come compromesso ottimale tra *accuracy* e praticità di implementazione su hardware embedded. I risultati ottenuti con le diverse tecniche di analisi hanno confermato la superiore *explainability* del modello **Fine Tree**, intrinsecamente interpretabile grazie alla sua struttura ad albero. Al contrario, il **SVM Cubic** presenta il limite di essere un modello black-box, risultando quindi non interpretabile.

I risultati complessivi del progetto evidenziano come l'approccio adottato, basato su modelli di classificazione supervisionata e sull'estrazione di feature significative sia nel dominio del tempo che della frequenza, consenta di ottenere un sistema diagnostico efficace per droni esarotori. L'elevata *accuracy* raggiunta dai modelli nella fase classificazione del guasto conferma la validità delle scelte metodologiche, mentre lo studio dell'*explainability* ha permesso di identificare modelli interpretabili e di comprendere meglio l'influenza delle caratteristiche dei dati sulle predizioni.

Per quanto riguarda la classificazione binaria, i risultati indicano che la distinzione tra condizioni nominali e anomalie può essere effettuata con elevata precisione. In questo contesto, il modello **Boosted Trees** ha mostrato le migliori prestazioni, confermando la sua efficacia nel rilevare rapidamente la presenza di guasti pur mantenendo una buona robustezza rispetto alla variabilità dei dati.

Dal punto di vista della classificazione ternaria, il modello **Fine Tree** rappresenta una soluzione interessante per l'implementazione su sistemi embedded, grazie alla sua interpretabilità nativa e al basso costo computazionale, senza compromettere significativamente le prestazioni di classificazione. L'analisi comparativa con il **SVM Cubic** ha evidenziato i limiti dei modelli black-box in contesti in cui trasparenza e tracciabilità delle decisioni sono requisiti fondamentali.

Il progetto dimostra come un approccio strutturato alla diagnosi dei guasti, integrando *preprocessing* accurato, estrazione di *feature* mirate, classificazione supervisionata e analisi dell'interpretabilità dei modelli, possa fornire un supporto efficace e affidabile alla manutenzione predittiva dei droni esarotori, sia a livello di rilevazione generale dei guasti sia di caratterizzazione specifica dei diversi tipi di degrado.

Bibliografia

[1] Alessandro Baldini, Lorenzo D'Alleva, Riccardo Felicetti, Francesco Ferracuti, Alessandro Freddi, and Andrea Monteriù. Uav-fd: a dataset for actuator fault detection in multirotor drones *. pages 998–1004, 2023.

[2] J. Cação, J. Santos, and M. Antunes. Explainable ai for industrial fault diagnosis: A systematic review. *Journal of Industrial Information Integration*, 47:100905, 2025.

[3] Trong-Du Nguyen, Thanh-Hai Nguyen, Danh-Thanh-Binh Do, Thai-Hung Pham, Jin-Wei Liang, and Phong-Dien Nguyen. Efficient and explainable bearing condition monitoring with decision tree-based feature learning. *Machines*, 13(6), 2025.