

*UNIVERSITÀ POLITECNICA DELLE MARCHE*

*FACOLTÀ DI INGEGNERIA*



*Corso di Laurea Magistrale in  
Ingegneria Informatica e dell'Automazione*

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

Professore:  
PROF. FREDDI ALESSANDRO

Sviluppatori:  
CAPORUSSO CHIARA AMALIA  
GALEAZZI MARGHERITA

ANNO ACCADEMICO 2022-2023

Il codice è disponibile al seguente indirizzo: <https://github.com/ChiaraAmalia/ProgettoManutenzionePreventiva>

Per eseguire il nostro progetto sarà sufficiente scaricare il contenuto del repository di GitHub, e poi mandare in run il file **analisi\_tutti\_15.ipynb**. Non è necessario mandare in run gli altri file in quanto i database relativi ai vari voli sono già stati pre-processati e sono presenti nella repository. Ciascun file relativo ad ogni volo per il calcolo delle feature è presente all'interno delle cartelle M1..M6, divise ognuna in ulteriori tre cartelle, relative ai log di volo considerati.

# Indice

<b>Indice</b>	<b>i</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Dataset . . . . .	1
<b>2 Strumenti e metodi</b>	<b>4</b>
2.1 Linguaggi e librerie . . . . .	4
2.2 Strumenti . . . . .	5
<b>3 Sviluppo del progetto</b>	<b>6</b>
3.1 Generazione dei file . . . . .	7
3.2 Costruzione del dataset . . . . .	10
3.2.1 Trimming del datalog . . . . .	10
3.2.2 Sincronizzazione dei tempi . . . . .	13
3.2.3 Estrazione delle feature nel tempo e in frequenza . . . . .	15
3.3 Classificazione . . . . .	18
3.3.1 Creazione del dataframe . . . . .	18
3.3.2 Bilanciamento del dataset . . . . .	19
3.3.3 Selezione delle feature . . . . .	20
3.3.4 Separazione del dataset . . . . .	22
3.3.5 Definizione dei classificatori . . . . .	22
3.3.6 Addestramento e validazione . . . . .	23
3.3.7 Matrici di confusione . . . . .	25
3.3.8 Cross Validation Scores . . . . .	26
3.3.9 Report Classificazione . . . . .	26
<b>4 Risultati</b>	<b>28</b>
4.1 Logistic Regression . . . . .	28
4.1.1 Curva ROC . . . . .	29
4.1.2 Matrice di confusione . . . . .	29
4.1.3 Report della classificazione . . . . .	30
4.2 Decision Tree . . . . .	30
4.2.1 Curva ROC . . . . .	31
4.2.2 Matrice di confusione . . . . .	31
4.2.3 Report della classificazione . . . . .	32

---

4.3	Random Forest . . . . .	32
4.3.1	Curva ROC . . . . .	33
4.3.2	Matrice di confusione . . . . .	33
4.3.3	Report della classificazione . . . . .	34
4.4	Support Vector Machine . . . . .	35
4.4.1	Curva ROC . . . . .	35
4.4.2	Matrice di confusione . . . . .	36
4.4.3	Report della classificazione . . . . .	37
4.5	Multi Layer Perceptron . . . . .	37
4.5.1	Curva ROC . . . . .	38
4.5.2	Matrice di confusione . . . . .	38
4.5.3	Report della classificazione . . . . .	39
4.6	Stochastic Gradient Descent . . . . .	39
4.6.1	Curva ROC . . . . .	40
4.6.2	Matrice di confusione . . . . .	41
4.6.3	Report della classificazione . . . . .	42
4.7	Cross-Validation Scores . . . . .	43
<b>5</b>	<b>Conclusioni e Sviluppi futuri</b>	<b>44</b>
	<b>Bibliografia</b>	<b>46</b>
	<b>Elenco delle figure</b>	<b>47</b>
	<b>Elenco delle tabelle</b>	<b>48</b>
	<b>Elenco dei codici</b>	<b>49</b>

# Capitolo 1

## Introduzione

Nella seguente relazione verrà illustrato lo svolgimento di questo progetto nell'ambito dell'individuazione di eventuali guasti, relativamente ad un drone esarotore in volo, a partire dall'analisi dei dati raccolti.

Si è iniziato con la raccolta e l'analisi dei dati ottenuti, andando ad analizzare le variabili e i parametri relativi ad esse, cercandone di capire l'utilità ai fini dei nostri scopi. Successivamente, si è proceduto con una fase di ETL andando ad individuare solamente i parametri utili ai fini dell'estrazione delle feature. Come ultimo passaggio, sono stati considerati i valori delle feature relativamente ai diversi casi considerati, 18 in tutto, considerando le diverse casistiche per ciascun motore del drone (nessun guasto, guasto relativo alla pala del drone esarotore usurata al 5%, guasto relativo alla pala del drone esarotore usurata del 10%). Nel seguito verranno illustrati i risultati ottenuti con il procedimento seguito.

### 1.1 Dataset

Il dataset è composto da diversi voli, ognuno dei quali in differenti condizioni operative: pala nuova, pala usurata ("tagliata" nella misura del 5% della sua lunghezza) e pala molto usurata ("tagliata" nella misura del 10% della sua lunghezza).

Dal datalog fornitoci siamo andate a recuperare solamente una parte delle variabili presenti al suo interno, ovvero quelle che abbiamo ritenuto più importanti al fine delle successive analisi. Di seguito riportiamo una breve descrizione delle variabili prese in esame e del relativo contenuto, recuperata dal sito di ardupilot[1].

In ciascuna tabella riportata è compreso anche il campo **TimeUS**, indicante l'istante di tempo (in microsecondi), in cui è stata rilevata la misurazione dei parametri riportati (i microsecondi partono dal momento in cui il sistema è stato avviato). Tale parametro è risultato poi essere di fondamentale importanza per la sincronizzazione dei tempi e l'unione di tutti i parametri utili ai fini delle analisi effettuate.

ATT (informazioni di attitudine)	
Parametro	Descrizione
DesRoll	Angolo desiderato di rollio dal pilota, in gradi (se negativo a sinistra, se positivo a destra)
Roll	Angolo di rollio effettivo del velivolo, in gradi (se negativo a sinistra, se positivo a destra)
DesPitch	Angolo di beccheggio desiderato dal pilota, in gradi (se negativo in avanti, se positivo all'indietro)
Pitch	Angolo di beccheggio effettivo del velivolo, in gradi (se negativo in avanti, se positivo all'indietro)
DesYaw	Direzione desiderata dal pilota, in gradi (0 = nord)
Yaw	Direzione effettiva del velivolo, in gradi (0 = nord)

Tabella 1.1: Informazioni di attitudine

I parametri presenti nella tabella 1.1 vengono considerati per poter prendere i desiderati e gli effettivi dei valori di Rollio, Beccheggio e Bardatura. Questi rappresentano gli angoli di inclinazione del velivolo, insieme ai valori desiderati dagli angoli imposti dal controllore, che vengono assegnati da radiocomando.

ESC (Feedback ricevuto dagli ESC)	
Parametro	Descrizione
Instance	Numero di istanza ESC, relativa al motore considerato
RPM	Tasso di rotazione del motore riportato
Curr	Corrente di input ricevuta dall'ESC

Tabella 1.2: Feedback ricevuto dagli ESC

L' Electronic Speed Controller (ESC)<sup>1.2</sup> è un componente fondamentale per il corretto funzionamento di un drone multirottore. La sua funzione è quella di collegare il controllore di volo con i motori consentendo la regolazione della velocità degli stessi. Ogni motore è dotato di un suo ESC perché in un sistema di volo multirottore, ogni motore avrà regimi di rotazione diversi rispetto agli altri. In RPM abbiamo i valori misurati dal sensore di velocità relativo al dato ESC, in CURR abbiamo la corrente di alimentazione dell'ESC. Il drone ha una sensorizzazione ad hoc sul motore.

IMU (Informazioni relative all'accelerometro e giroscopio)	
Parametro	Descrizione
Instance	Numero di istanza IMU
GyrX, GyrY, GyrZ	Velocità di rotazione grezza del giroscopio in rad/sec
AccX, AccY, AccZ	Valori grezzi dell'accelerometro in $m/s^2$

Tabella 1.3: Informazioni relative all'accelerometro e giroscopio

I parametri presenti nella tabella 1.3 vengono presi per tenere in considerazione la velocità angolare del giroscopio e i relativi valori dell'accelerometro. Tale variabile contiene i dati ad alta frequenza, intorno ai 350 hz.

RCOU (Valori di uscita servo channel da 9 a 14)	
Parametro	Descrizione
C9,...,C14	Output canali considerati

Tabella 1.4: Valori di uscita servo channel da 9 a 14

I valori riportati nelle variabili del parametro RCOU 1.4, contengono i comandi di Pulse-With Modulation (Modulazione della larghezza di impulso) relativi ai sei motori, in cui i canali attivi sono da C9 a C14.

XKF (Output stimatore EKF3)	
Parametro	Descrizione
Roll	Rollio stimato
Pitch	Beccheggio stimato
Yaw	Imbardata stimata

Tabella 1.5: Informazioni di attitudine

L'obiettivo nell'utilizzo della variabile XKF, riportata nella tabella 1.5 è verificare che i valori di Roll, Pitch e Yaw siano gli stessi di quelli presenti nella variabile ATT. XKF è una misura aggregata che viene generata da un topic, che si occupa di compiere un filtraggio alla Kalman.

# Capitolo 2

## Strumenti e metodi

### 2.1 Linguaggi e librerie

Tutto il codice presente nel progetto è stato scritto in linguaggio Python[2], sfruttando svariate librerie legate al mondo della rappresentazione dei dati, del calcolo scientifico e della classificazione. Tra le più importanti possiamo individuare:

- **Pandas:** è uno strumento per la manipolazione e l'analisi dei dati veloce e semplice da utilizzare.
- **SciPy:** è una libreria open source di algoritmi e strumenti matematici. Contiene moduli per l'ottimizzazione, per l'algebra lineare, l'integrazione, funzioni speciali, FFT, elaborazione di segnali ed immagini, solver ODE e altri strumenti comuni nelle scienze e nell'ingegneria.
- **Scikit-Learn:** è una libreria open source di apprendimento automatico. Contiene algoritmi di classificazione, regressione e clustering (raggruppamento) e macchine a vettori di supporto, regressione logistica, classificatore bayesiano, k-mean e DBSCAN, ed è progettato per operare con le librerie NumPy e SciPy.
- **Imbalanced-Learn:** è una libreria open-source che si basa su Scikit-Learn che fornisce strumenti per gestire la classificazione con classi sbilanciate
- **Seaborn:** Seaborn è una libreria in Python utilizzata principalmente per creare grafici statistici.

Dato che il datalog di partenza ci è stato fornito in formato .mat, per un totale di 18 file, è risultato necessario l'utilizzo di MATLAB[3], piattaforma di programmazione e calcolo numerico, al fine di estrarre solamente le variabili utili ai fini dell'analisi, trasformandole in file .csv.



## 2.2 Strumenti

Per lo sviluppo di questo progetto, è necessario installare i seguenti tool:

- Installazione di MATLAB per l'estrazione dei file utili ai fini dell'analisi.
- Installazione della versione Python 3.10.2, utilizzata per lo sviluppo del progetto.
- Installazione delle librerie Python utilizzate: Pandas, SciPy, Scikit-Learn, Imbalanced-Learn Seaborn.
- Installazione di Jupyter Notebook[4], per l'elaborazione interattiva in tutti i linguaggi di programmazione, utilizzato per l'elaborazione di parti di codice al fine di analizzarne i risultati intermedi.

# Capitolo 3

## Sviluppo del progetto

L'obiettivo del progetto consiste nell'analizzare i dataset di volo di un drone esattore in caso di pale nuove e pale usurate (i.e. una pala appositamente danneggiata) al fine di discriminare il funzionamento.

Si è partiti anzitutto andando ad analizzare le variabili e i relativi parametri presenti nel datalog e, essendo questo riportato per ciascun motore del drone, per ognuna delle casistiche considerate (nessun guasto, guasto al 5%, guasto al 10%), in un file con estensione .mat, è risultato necessario l'utilizzo di MATLAB per l'estrazione delle variabili di interesse.

Successivamente, si è proceduto con il calcolo delle feature in Python, con un'importante fase di pre-processing volta al trimming e alla sincronizzazione dei tempi con tutte le variabili considerate.

Infine, con le feature calcolate nel tempo e in frequenza, si è proceduto con l'analisi vera e propria dei dati, mediante la costruzione di un modello in grado di predire, su una finestra temporale di un secondo, se il drone considerato nel volo presenta un guasto di una certa entità sopra riportata o meno.

## 3.1 Generazione dei file

Inizialmente, il datalog a disposizione era sottoforma di file .mat, quindi manipolabile solamente mediante l'utilizzo dell'ambiente software MATLAB. Essendo il datalog, per ciascun volo considerato, una tabella composta da un certo numero di matrici, l'obiettivo era quello di creare tanti file .csv, contenenti ognuno una matrice (ognuna associata a una variabile considerata) con i relativi valori, ognuno con la propria intestazione. Per ciascun motore e, di conseguenza, ciascun log di volo fornitoci, abbiamo creato delle cartelle, ognuna contenente il file .mat associato a quel log 3.1.

```
.  
├─ M1  
│   ├── FAULT_M1_10  
│   │   └─ FAULT_M1_10%.mat  
│   ├── FAULT_M1_5  
│   │   └─ FAULT_M1_5%.mat  
│   └─ NO_FAULT1  
│       └─ NO_FAULT1.mat  
├─ M2  
│   ├── FAULT_M2_10  
│   │   └─ FAULT_M2_10%.mat  
│   ├── FAULT_M2_5  
│   │   └─ FAULT_M2_5%.mat  
│   └─ NO_FAULT2  
│       └─ NO_FAULT2.mat  
├─ M3  
│   ├── FAULT_M3_10  
│   │   └─ FAULT_M3_10%.mat  
│   ├── FAULT_M3_5  
│   │   └─ FAULT_M3_5%.mat  
│   └─ NO_FAULT3  
│       └─ NO_FAULT3.mat  
├─ M4  
│   ├── FAULT_M4_10  
│   │   └─ FAULT_M4_10%.mat  
│   ├── FAULT_M4_5  
│   │   └─ FAULT_M4_5%.mat  
│   └─ NO_FAULT4  
│       └─ NO_FAULT4.mat  
├─ M5  
│   ├── FAULT_M5_10  
│   │   └─ FAULT_M5_10%.mat  
│   ├── FAULT_M5_5  
│   │   └─ FAULT_M5_5%.mat  
│   └─ NO_FAULT5  
│       └─ NO_FAULT5.mat  
└─ M6  
    ├── FAULT_M6_10  
    │   └─ FAULT_M6_10%.mat  
    ├── FAULT_M6_5  
    │   └─ FAULT_M6_5%.mat  
    └─ NO_FAULT6  
        └─ NO_FAULT6.mat
```

Figura 3.1: Lista delle cartelle che contengono il relativo file .mat

Successivamente, si è proceduto al caricamento in MATLAB di ciascun file, in modo da poterne analizzare le variabili al suo interno. Ciascuna variabile presente all'interno di ogni file viene rappresentata per mezzo di una matrice, così come la relativa intestazione, composta da un certo numero di righe ed una sola colonna

### 3.2.

ATT												ATT_label	
10169x11 double												11x1 cell	
	1	2	3	4	5	6	7	8	9	10	11	1	2
1	412	45015573	0	0.8600	0	-0.7200	23.2400	23.2300	0	0	3	LineNo	
2	869	45100988	0	0.8800	0	-0.7300	23.2500	23.2400	0	0	3	TimeUS	
3	1096	45179414	0	0.8600	0	-0.7300	23.2500	23.2400	0	0	3	DesRoll	
4	1298	45306143	0	0.8600	0	-0.7200	23.2600	23.2500	0	0	3	Roll	
5	1551	45344573	0	0.8600	0	-0.7100	23.2600	23.2500	0	0	3	DesPitch	
6	1574	45364873	0	0.8600	0	-0.7100	23.2600	23.2500	0	0	3	Pitch	
7	1616	45375143	0	0.8600	0	-0.7100	23.2600	23.2500	0	0	3	DesYaw	
8	1657	45384873	0	0.8600	0	-0.7100	23.2600	23.2500	0	0	3	Yaw	
9	1697	45394932	0	0.8600	0	-0.7100	23.2600	23.2500	0	0	3	ErrRP	
10	1738	45404708	0	0.8600	0	-0.7100	23.2600	23.2400	0	0	3	ErrYaw	
11	1780	45415443	0	0.8600	0	-0.7100	23.2500	23.2400	0	0	3	11 AEKF	
12	1822	45425532	0	0.8600	0	-0.7200	23.2600	23.2400	0	0	3	12	
13	1862	45435993	0	0.8600	0	-0.7200	23.2500	23.2400	0	0	3	13	
14	1903	45446496	0	0.8600	0	-0.7200	23.2500	23.2400	0	0	3	14	
15	1948	45456938	0	0.8600	0	-0.7200	23.2500	23.2400	0	0	3	15	
16	1988	45466595	0	0.8600	0	-0.7200	23.2500	23.2400	0	0.0100	3	16	
17	2031	45476516	0	0.8600	0	-0.7200	23.2500	23.2400	0	0.0100	3	17	
18	2071	45487390	0	0.8600	0	-0.7200	23.2500	23.2400	0	0.0100	3	18	
19	2115	45500657	0	0.8600	0	-0.7100	23.2500	23.2400	0	0.0100	3	19	
20	2155	45510869	0	0.8600	0	-0.7100	23.2500	23.2400	0	0.0100	3	20	
21	2196	45520917	0	0.8600	0	-0.7100	23.2500	23.2400	0	0.0100	3	21	

Figura 3.2: Rappresentazione matriciale ed intestazione relativa alla variabile ATT

A questo punto si è proceduto con la creazione del file .csv, contenenti tali matrici e relativi valori.

```

1 %caricamento del file .mat
2 load 'FAULT_M2_5%.mat'
3 %creazione del file .csv
4 fid = fopen('ATT.csv','w');
5 %scrittura dell'intestazione sul file
6 textHeader = strjoin(ATT_label, ',');
7 fprintf(fid,'%s\n',textHeader);
8 fclose(fid);
9 %scrittura della matrice in corrispondenza dell'intestazione
10 dlmwrite('ATT.csv',ATT,'precision',16,'-append','delimiter',',');

```

Codice 3.1: Script per convertire in file .csv le matrici presenti nei file .mat

Tale codice è stato utilizzato per la creazione dei file .csv utilizzati nelle successive analisi. Il file .mat viene anzitutto caricato nell'ambiente software, in modo da permetterci l'accesso alle svariate matrici presenti 3.3.















	ACC_label	7x1 cell
	ADSB_label	10x1 cell
	AHR2	10162x12 double
	AHR2_label	12x1 cell
	AIS1_label	17x1 cell
	AIS4_label	16x1 cell
	AIS5_label	17x1 cell
	AISR_label	6x1 cell
	AOA_label	4x1 cell
	ARM	4x6 double
	ARM_label	6x1 cell
	ARSP_label	13x1 cell
	ATT	10169x11 double
	ATT_label	11x1 cell

Figura 3.3: Workspace contenente tutte le matrici e relative intestazioni

A questo punto, si è proceduto alla creazione del file che ospiterà ciascuna matrice, andando ad inserire prima l'intestazione relativa a quella variabile e, successivamente i valori numerici, contenuti all'interno della corrispondente matrice. Nella costruzione del file si è dovuto specificare la lunghezza massima relativamente a ciascuna cella (16) in quanto alcuni valori venivano tagliati e quindi approssimati, rendendo inutilizzabile tale campo numerico. Si è scelta poi la virgola come delimitatore tra un campo e l'altro in quanto di più semplice integrazione all'interno dell'ambiente Python. Per ciascun file .mat, vengono generati un totale di 13 file, relativamente alle variabili considerate 3.4:

```

.
├─ ATT.csv
├─ ESC_0.csv
├─ ESC_1.csv
├─ ESC_2.csv
├─ ESC_3.csv
├─ ESC_4.csv
├─ ESC_5.csv
├─ IMU_0.csv
├─ IMU_1.csv
├─ IMU_2.csv
├─ RCOU.csv
├─ XKF1_0.csv
└─ XKF1_1.csv

```

Figura 3.4: Elenco file generati

Ciascun file si presenta nel modo riportato in Figura 3.5:

```
LineNo,TimeUS,DesRoll,Roll,DesPitch,Pitch,DesYaw,Yaw,ErrRP,ErrYaw,AEKF
412,45015573,0,0.86,0,-0.72,23.24,23.23,0,0,3
869,45100998,0,0.88,0,-0.73,23.25,23.24,0,0,3
1096,45179414,0,0.86,0,-0.73,23.25,23.24,0,0,3
1298,45306143,0,0.86,0,-0.72,23.26,23.25,0,0,3
1551,45344573,0,0.86,0,-0.71,23.26,23.25,0,0,3
1574,45364873,0,0.86,0,-0.71,23.26,23.25,0,0,3
1616,45375143,0,0.86,0,-0.71,23.26,23.25,0,0,3
1657,45384873,0,0.86,0,-0.71,23.26,23.25,0,0,3
1697,45394932,0,0.86,0,-0.71,23.26,23.25,0,0,3
1738,45404708,0,0.86,0,-0.71,23.26,23.24,0,0,3
1780,45415443,0,0.86,0,-0.71,23.25,23.24,0,0,3
1822,45425532,0,0.86,0,-0.72,23.26,23.24,0,0,3
1862,45435993,0,0.86,0,-0.72,23.25,23.24,0,0,3
1903,45446496,0,0.86,0,-0.72,23.25,23.24,0,0,3
1948,45456938,0,0.86,0,-0.72,23.25,23.24,0,0,3
1988,45466595,0,0.86,0,-0.72,23.25,23.24,0,0.01,3
2031,45476516,0,0.86,0,-0.72,23.25,23.24,0,0.01,3
2071,45487390,0,0.86,0,-0.72,23.25,23.24,0,0.01,3
2115,45500657,0,0.86,0,-0.71,23.25,23.24,0,0.01,3
2155,45510869,0,0.86,0,-0.71,23.25,23.24,0,0.01,3
2196,45520917,0,0.86,0,-0.71,23.25,23.24,0,0.01,3
2240,45531545,0,0.86,0,-0.71,23.24,23.23,0,0.01,3
2281,45541515,0,0.86,0,-0.71,23.24,23.23,0,0.01,3
2322,45551445,0,0.86,0,-0.71,23.24,23.23,0,0.01,3
2368,45561542,0,0.86,0,-0.71,23.24,23.23,0,0.01,3
```

Figura 3.5: File csv generato

## 3.2 Costruzione del dataset

Avendo ottenuto i vari file .csv relativi a ciascun volo, procediamo con il calcolo delle feature ai fini della classificazione. All'interno di ciascuna cartella in cui sono contenuti i file csv relativi al volo considerato, è presente un file con estensione .ipynb che ci consente di costruire il dataset finale contenente tutte le feature.

### 3.2.1 Trimming del datalog

Tutti le acquisizioni effettuate nei vari istanti di tempo partono dal momento in cui il drone è stato "armato" ovvero nel momento in cui le pale dei vari motori iniziano a ruotare. È risultato quindi necessario andare ad eliminare la parte iniziale e finale in cui il drone è acceso ma non è ancora decollato. Sono parecchie le acquisizioni con un valore costante che corrisponde al minimo, ovvero quando il drone viene acceso ma gli si dà una potenza minima che non gli permette di decollare. Andiamo quindi a togliere tutti quei valori prima di arrivare ad una potenza e quindi un'altezza minima e quelli successivi al momento in cui il drone sta per atterrare, andando quindi a prendere solamente l'arco di tempo in cui il drone sta volando.

### Variabile RCOU

In tale variabile sono presenti i canali di output relativi alla potenza dei sei motori. Questa variabile è risultata di fondamentale importanza in quanto è servita per effettuare il trimming del datalog.

---

```

1 #andiamo a prendere una potenza media di 1350 per considerare il
  momento il cui inizia/finisce il volo
2 rcou_m1 = pd.read_csv("RCOU.csv")
3 rcou_m1 = rcou_m1.drop(['LineNo', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6',
  'C7', 'C8'], axis=1)
4 potenza = 1350
5 rcou_m1 = rcou_m1[((rcou_m1['C9'] >= potenza) & (rcou_m1['C10'] >=
  potenza) & (rcou_m1['C11'] >= potenza) & (rcou_m1['C12'] >=
  potenza) & (rcou_m1['C13'] >= potenza) & (rcou_m1['C14'] >=
  potenza))]
6 rcou_m1 = rcou_m1.reset_index(drop=True)

```

---

Codice 3.2: Estrazione dati RCOU

Dopo aver recuperato il file .csv relativo alla variabile RCOU, siamo andati a rimuovere tutti quei canali che non risultavano essere attivi, andando quindi a considerare solo dal C9 al C14. Abbiamo considerato una soglia di potenza dei motori media pari a 1350 in quanto, se la potenza dei motori è pari a 1000 allora significa che non passa corrente quindi il drone ancora non è in volo inoltre, la potenza media massima che viene raggiunta è intorno ai 1700; di conseguenza, 1350 risulta la scelta migliore. Siamo quindi andati a prendere tutti i valori tali per cui la potenza in quei canali era maggiore o uguale a 1350.

---

```

1 min=rcou_m1['TimeUS'][0]
2 max=rcou_m1['TimeUS'][len(rcou_m1)-1]

```

---

Codice 3.3: Preso tempo di inizio e tempo di fine del volo

A questo punto, dal dataframe ottenuto andiamo a prendere i valori di TimeUS ovvero degli istanti di tempo minimo e massimo per effettuare il trimming con le altre variabili, al fine di considerare solamente la finestra temporale in cui il drone è effettivamente in volo.

### Confronto tra le variabili ATT e XKF

A questo punto andiamo a prendere i parametri relativi alle variabili ATT e XKF per controllare che i valori di Roll, Pitch e Yaw siano effettivamente gli stessi.

---

```

1 #variabile ATT
2 att_m1 = pd.read_csv("ATT.csv")
3 att_m1 = att_m1.drop(['LineNo', 'ErrRP', 'ErrYaw', 'AEKF'], axis=1)
4 att_m1 = att_m1[((att_m1['TimeUS'] >= min) & (att_m1['TimeUS'] <=
  max))]
5 att_m1 = att_m1.reset_index(drop=True)
6
7 #variabile XKF

```

---

---

```

8 xkf1_m1 = pd.read_csv("XKF1_0.csv")
9 xkf1_m1 = xkf1_m1.drop(['LineNo', 'C', 'VN', 'VE', 'VD', 'dPD', 'PN', 'PE',
    ', 'PD', 'GX', 'GY', 'GZ', 'OH'], axis=1)
10 xkf1_m1 = xkf1_m1[((xkf1_m1['TimeUS'] >= min) & (xkf1_m1['TimeUS']
    <= max))]
11 xkf1_m1 = xkf1_m1.reset_index(drop=True)

```

---

Codice 3.4: Estrazione dati ATT e XKF

Siamo quindi andati a prendere le due variabili di interesse, rimuovere i parametri non utili ai fini dell'analisi e togliere la parte del datalog che non rientra nella finestra temporale considerata.

---

```

1 for idx,i in enumerate(att_m1['Roll']):
2     if(xkf1_m1['Roll'][idx] != i):
3         att_m1['Roll'][idx] = ((att_m1['Roll'][idx] + xkf1_m1['
    Roll'][idx])/2)
4
5 for idx,i in enumerate(att_m1['Pitch']):
6     if(xkf1_m1['Pitch'][idx] != i):
7         att_m1['Pitch'][idx] = ((att_m1['Pitch'][idx] + xkf1_m1['
    Pitch'][idx])/2)
8
9 for idx,i in enumerate(att_m1['Yaw']):
10    if(xkf1_m1['Yaw'][idx] != i):
11        att_m1['Yaw'][idx] = ((att_m1['Yaw'][idx] + xkf1_m1['Yaw'
    '][idx])/2)

```

---

Codice 3.5: Confronto valori ATT e XFK

Abbiamo quindi proceduto al confronto dei valori nei parametri delle variabili per vedere se effettivamente corrispondessero. Se tali valori non corrispondono viene effettuata una media tra i due, andando a riassegnare il valore alla variabile ATT in corrispondenza di quel parametro.

### Variabili ESC

A questo punto procediamo con l'elettricità che scorre nei motori. In questo caso, sono sei le istanze di ESC che consideriamo, una per ogni motore del drone. Andiamo anzitutto a rimuovere tutti parametri non utili ai fini della nostra analisi, tenendo quindi in considerazione solamente i parametri relativi ad RPM e Curr, ritenuti i più importanti ed effettuando il trimming di tali parametri, prendendo solamente la finestra temporale compresa nell'intervallo di tempo tra i valori di min e max calcolati precedentemente.

### Variabili IMU

Per quanto riguarda le variabili IMU, sono tre le istanze che abbiamo a disposizione. Dopo aver rimosso tutti i parametri non utili ai fini delle analisi che si vogliono effettuare, per ogni variabile si è presa solamente la finestra temporale compresa nell'intervallo di tempo tra i valori di min e max calcolati precedentemente. Al



fine di considerare solamente una variabile associata all'IMU, si è optato prima per una concatenazione di queste con un successivo calcolo della media, effettuando un raggruppamento sulla colonna dei tempi.

---

```

1 imu_0_m1 = pd.read_csv("IMU_0.csv")
2 imu_0_m1 = imu_0_m1.drop(['LineNo', 'I', 'EG', 'EA', 'T', 'GH', 'AH',
3                             'GHZ', 'AHZ'], axis=1)
4 imu_0_m1 = imu_0_m1[((imu_0_m1['TimeUS'] >= min) & (imu_0_m1['TimeUS'] <= max))]
5
6 imu_1_m1 = pd.read_csv("IMU_1.csv")
7 imu_1_m1 = imu_1_m1.drop(['LineNo', 'I', 'EG', 'EA', 'T', 'GH', 'AH',
8                             'GHZ', 'AHZ'], axis=1)
9 imu_1_m1 = imu_1_m1[((imu_1_m1['TimeUS'] >= min) & (imu_1_m1['TimeUS'] <= max))]
10
11 imu_2_m1 = pd.read_csv("IMU_2.csv")
12 imu_2_m1 = imu_2_m1.drop(['LineNo', 'I', 'EG', 'EA', 'T', 'GH', 'AH',
13                             'GHZ', 'AHZ'], axis=1)
14 imu_2_m1 = imu_2_m1[((imu_2_m1['TimeUS'] >= min) & (imu_2_m1['TimeUS'] <= max))]
15
16 imu_m1 = pd.concat((imu_0_m1, imu_1_m1, imu_2_m1))
17 imu_m1 = imu_m1.groupby(imu_m1.TimeUS, as_index=False).mean()

```

---

Codice 3.6: Estrazione dati IMU

Ciò che si ottiene quindi è un'unica variabile contenente al suo interno una media delle tre variabili IMU.

### 3.2.2 Sincronizzazione dei tempi

In questa fase, avendo ottenuto tutte le variabili e i relativi parametri utili ai fini delle analisi, è risultato necessario, per poter unire tutte le tabelle, effettuare una sincronizzazione dei tempi.

#### Unione delle tabelle

Il primo passo di questa fase è stato unire tutte le tabelle, risultanti dalle precedenti operazioni. Per fare ciò abbiamo sfruttato il metodo di pandas *merge\_ordered()*, progettato per dati ordinati come dati di serie temporali e che semplicemente esegue l'unione di questi dati (mantenendo l'ordine).

In seguito abbiamo proceduto andando ad imputare i valori mancanti e per fare ciò abbiamo utilizzato un altro metodo di pandas, ovvero *fillna()* che permette di andare a riempire i valori mancanti con il metodo specificato tra le parentesi.

Come è possibile vedere nel codice riportato sotto, noi abbiamo applicato tale funzione due volte, una prima volta con il metodo *"ffill"* che sostanzialmente propaga l'ultima osservazione valida in avanti fino alla prossima valida, e successivamente, per riempire i valori che in questo modo rimanevano vuoti, con il metodo *"bfill"* che utilizza l'osservazione valida successiva per colmare le lacune (lavora all'indietro).

---

```

1 m1_nofault = pd.merge_ordered(imu_m1, att_m1)
2 m1_nofault = m1_nofault.fillna(method="ffill").fillna(method="bfill")
3 m1_nofault = pd.merge_ordered(m1_nofault, rcou_m1)
4 m1_nofault = m1_nofault.fillna(method="ffill").fillna(method="bfill")

```

---

Codice 3.7: Unione delle tabelle

Come si vede nel codice sopra riportato, le variabili relative agli ESC non sono state aggiunte nel merge, in quanto la frequenza di campionamento di quest'ultimi era di diversi ordini di grandezza differente, comportando la ripetizione, per un numero molto grande di righe, del medesimo valore. Tale disagio ha poi portato ad ottenere, in fase di classificazione, per alcuni classificatori, un'accuratezza del 100%.

Dopo aver unito tutti i parametri delle tabelle, si è proceduto con la sincronizzazione dei tempi del volo, impostando a 0  $\mu s$  la prima riga della tabella, indicando il fatto che il volo inizi in quell'istante di tempo, andando quindi a sottrarre a ciascuna riga, relativamente alla colonna dei tempi, il valore della prima.

---

```

1 m1_nofault["TimeUS"] = m1_nofault["TimeUS"] - m1_nofault.iloc[0]["TimeUS"]
2 m1_nofault["TimeUS"] = m1_nofault["TimeUS"].astype(int)

```

---

Codice 3.8: Settaggio istante di partenza pari a 0

### Sincronizzazione con 350 misurazioni al secondo

Essendo noto il fatto che il campionamento avveniva con una frequenza pari a:  $f_c = 350$ , ma che i sensori potevano in realtà campionare a tempi leggermente diversi, si è forzatamente sincronizzato il dataset per fare in modo che le misurazioni fossero effettivamente 350 al secondo.

Quindi noto che il tempo di campionamento fosse pari a:  $T = 1/f_c = 1/350 = 0.002857s = 2857\mu s$ , utilizzando un semplice ciclo *for*, si è creato un dataframe con i tempi in modo tale che tra le varie misurazioni trascorressero veramente  $2857\mu s$ . Infine si è unito questo dataframe a quello precedentemente ottenuto, andando a rimuovere le righe con i tempi che non rispettavano la distanza tra le misurazioni.

---

```

1 last_cell = m1_nofault.iloc[-1, m1_nofault.columns.get_loc('TimeUS')]
2
3 to_be = []
4 for i in range(0, last_cell, 2857):
5     to_be.append(i)
6
7 df = pd.DataFrame(to_be, columns = ['TimeUS'])
8
9 m1_nofault_final = pd.merge_ordered(m1_nofault, df)
10 m1_nofault_final = m1_nofault_final.fillna(method="ffill").fillna(
    method="bfill")

```

---

---

```

11
12 m1_nofault_final = m1_nofault_final[m1_nofault_final.TimeUS.isin(
    to_be)]

```

---

Codice 3.9: Sincronizzazione con 350 misurazioni al secondo

A questo punto trasformiamo la colonna dei tempi, espressa in  $\mu\text{s}$ , in un formato HH:MM:SS.

---

```

1 m1_nofault_final=m1_nofault_final.reset_index(drop=True)
2 m1_nofault_final['TimeUS'] = pd.to_datetime(m1_nofault_final['
    TimeUS'], unit='us').dt.strftime('%H:%M:%S.%f')

```

---

Codice 3.10: Formattazione del tempo

Questa trasformazione la effettuiamo per verificare se effettivamente vi sono 2857  $\mu\text{s}$  tra una misurazione e l'altra, e per rendere più leggibile la variabile temporale.

### 3.2.3 Estrazione delle feature nel tempo e in frequenza

Dopo aver verificato se la sincronizzazione dei tempi è andata a buon fine, si è proceduto con il calcolo delle feature nel tempo ed in frequenza per vedere se queste sono significative.

Come prima cosa si è dovuta definire una funzione per il calcolo del Root Mean Square in quanto essa non era già implementata in pandas.

---

```

1 #Funzione che calcola il Root Mean Square (RMS)
2 def rmsValue(arr, n):
3     square = 0
4     mean = 0.0
5     root = 0.0
6     #Calcola il quadrato
7     for i in range(0,n):
8         square += (arr[i]**2)
9     #Calcola la media
10    mean = (square / (float)(n))
11    #Calcola la radice
12    root = math.sqrt(mean)
13    return root

```

---

Codice 3.11: Definita funzione per il calcolo del Root Mean Square

Successivamente si è proceduto creando una funzione che per ogni secondo andasse a calcolare:

- Feature nel tempo, quali:
  - media;
  - varianza;
  - curtosi;

- root mean square.
- Feature in frequenza, quali:
  - frequenza del 1°picco;
  - ampiezza del 1°picco;
  - frequenza del 2°picco;
  - ampiezza del 2°picco.

Per fare ciò si è utilizzata la funzione per il calcolo del root mean square introdotta in precedenza, e funzioni preimpostate per il calcolo delle altre feature. Si è poi trasformata la serie temporale in frequenza e si sono trovate le features in questo dominio.

### Funzione per il calcolo delle feature nel tempo ed in frequenza

---

```

1 def time_freq_feat(V):
2     N=350
3     height_threshold=0.0
4     tab= pd.DataFrame([], columns=['tempo',f'<>({V})',f'var({V})',
5     f'kurt({V})',f'RMS({V})',f'freq1({V})',f'freq2({V})',f'amp1({V}
6     )',f'amp2({V})'])
7     for i in range(int(len(m1_nofault_final)/350)):
8         V1=m1_nofault_final[str(V)].iloc[:((i+1)*350)+1]
9         mean=V1.mean()
10        var=V1.var()
11        kurt=V1.kurtosis()
12        rms=rmsValue(V1,len(V1))
13        yf=fft(V1.values)
14        xf=fftfreq(N,1/350)
15        peaks_index, properties = find_peaks(yf,height=-200,
16        prominence=0.1)
17
18        h_max_peak_1=np.unique(properties['peak_heights'])[-1]
19        peak1_dimension = np.where(properties['peak_heights'] ==
20        h_max_peak_1)
21        if (len(peak1_dimension[0])>1):
22            index1 = np.where(properties['peak_heights'] ==
23            h_max_peak_1)[0][1]
24        else:
25            index1 = np.where(properties['peak_heights'] ==
26            h_max_peak_1)[0][0]
27
28        h_max_peak_2=np.unique(properties['peak_heights'])[-2]
29        peak2_dimension = np.where(properties['peak_heights'] ==
30        h_max_peak_2)
31        if (len(peak2_dimension[0])>1):
32            index2 = np.where(properties['peak_heights'] ==
33            h_max_peak_2)[0][1]

```

---

```

27         else:
28             index2 = np.where(properties['peak_heights'] ==
h_max_peak_2)[0][0]
29
30             val={ 'tempo':i,f'<>({V})':mean,f'var({V})':var,f'kurt({V})
':kurt,f'RMS({V})':rms,f'freq1({V})':xf[index1],f'freq2({V})':
xf[index2],f'amp1({V})':h_max_peak_1,f'amp2({V})':h_max_peak_2}
31             tab=tab.append(val,ignore_index=True)
32         return tab

```

---

Codice 3.12: Definizione funzione per il calcolo delle feature nel tempo e in frequenza

### Creazione del dataset

In questa fase si è proceduto con l'applicazione della funzione precedentemente definita a tutte le variabili presenti all'interno della tabella sincronizzata.

---

```

1 GyrX=time_freq_feat('GyrX')
2 GyrY=time_freq_feat('GyrY')
3 GyrZ=time_freq_feat('GyrZ')
4 AccX=time_freq_feat('AccX')
5 AccY=time_freq_feat('AccY')
6 AccZ=time_freq_feat('AccZ')
7 Roll=time_freq_feat('Roll')
8 Pitch=time_freq_feat('Pitch')
9 Yaw=time_freq_feat('Yaw')
10 c_9 =time_freq_feat('C9')
11 c_10 =time_freq_feat('C10')
12 c_11 =time_freq_feat('C11')
13 c_12 =time_freq_feat('C12')
14 c_13 =time_freq_feat('C13')
15 c_14 =time_freq_feat('C14')

```

---

Codice 3.13: Calcolo feature per ogni variabile

Andiamo adesso ad effettuare un merge di tutte le tabelle ottenute dall'applicazione della funzione, sulla base della colonna dei tempi. Procediamo quindi con l'inserimento della colonna relativa al guasto associando, per quel datalog di volo che stiamo considerando, il guasto corrente.

---

```

1 merged = [GyrX,GyrY,GyrZ,AccX,AccY,AccZ,Roll,Pitch,Yaw,c_9,c_10,
c_11,c_12,c_13,c_14]
2 df_merged = reduce(lambda left,right: pd.merge(left,right,on=['
tempo'],how='outer'), merged)
3 df_merged["Guasto"] = guasto

```

---

Codice 3.14: Creazione del dataset

Il dataset contenente tutte le feature calcolate viene quindi convertito in formato .csv, per essere poi analizzato nella fase successiva

---

```

1 path_file = path_file.replace(r"M1\NO_FAULT1", "")

```

---

```
2 os.chdir(path_file)
3 df_merged.to_csv('m1_nofault.csv', index=False)
```

---

Codice 3.15: Salvataggio in .csv del dataset

## 3.3 Classificazione

Dopo aver ottenuto, per ciascun volo considerato, un file .csv ad esso associato contenente tutte le feature calcolate relativamente a quel volo, procediamo con la fase di addestramento di un modello in grado di predire la classe di guasto di un volo, a partire dall'analisi dei valori delle feature calcolate su una finestra temporale di un secondo.

### 3.3.1 Creazione del dataframe

Anzitutto, per poter effettuare una classificazione sui dati, è necessario che questi vengano riportati all'interno di un unico contenitore. Andremo quindi a creare un dataframe che al suo interno conterrà tutti i valori delle feature calcolate per ciascun volo preso in considerazione in questa analisi.

---

```
1 path_file = os.path.abspath(os.getcwd()) #prendiamo il path in cui
    si trova il file su cui stiamo lavorando
2 print(path_file)
3 os.chdir(path_file) #cambiamo directory al fine di poter prendere
    i file csv per la creazione del dataframe
4
5 data_path = os.path.join(path_file, '*.csv') #lista di tutti gli
    elementi con estensione csv nella cartella path_file
6 csv_list = glob.glob(data_path) #converte data path in un output
    Unix-like (ls) (*.csv -> lista di elementi con estensione csv)
7 df_1 = pd.DataFrame() #creazione di un dataframe vuoto
8
9 # ciclo per scorrere tutti i csv
10 for csv_file in csv_list: # ciclo che scorre i csv nella cartella
    path
11     df = pd.read_csv(csv_file)
12     df_1 = df_1.append(df, ignore_index=True)
```

---

Codice 3.16: Creazione del Dataframe

Prima di procedere con tale operazione, è risultato necessario spostarsi nella cartella in cui si trova il file in esame su cui è stata fatta classificazione. Abbiamo quindi estratto il path assoluto di tale file e cambiato quindi directory. Successivamente, è stata creata una lista contenente tutti gli elementi con estensione .csv nella cartella in esame. L'output risultante viene quindi trasformato in un output Unix-like.

Creiamo quindi un dataframe vuoto in cui andremo a mettere i nostri dati: scorriamo quindi la lista dei file csv presenti nella cartella e salviamoli all'interno del nostro dataframe.

### Rimozione duplicati

Al fine di poter effettuare una corretta classificazione, si è proceduto al controllo della presenza di eventuali duplicati all'interno del nostro dataframe.

---

```
1 df_1=df_1.drop_duplicates()
```

---

Codice 3.17: Rimozione di eventuali duplicati

Avendo rimosso a monte i dati relativi ai file ESC, non sono presenti duplicati all'interno del nostro dataframe.

### Shuffling

Si è proceduto quindi ad una fase di shuffling del dataframe al fine di ottenere una classificazione migliore.

---

```
1 df_1 = df_1.sample(frac=1).reset_index(drop=True)
```

---

Codice 3.18: Shuffling del dataframe

### Rimozione della colonna relativa al tempo

Al fine della classificazione, la colonna relativa al tempo non è risultata necessaria, in quanto l'uso di quest'ultima è risultato di particolare importanza per il trimming del datalog, la sincronizzazione dei tempi e il calcolo delle feature nella finestra temporale di un secondo, ma non ha la medesima importanza per il task di classificazione.

---

```
1 df_1 = df_1.drop(['tempo'], axis=1)
```

---

Codice 3.19: Rimozione della colonna "tempo"

Avendo tolto la colonna relativa al tempo siamo quindi rimasti con i valori delle feature e la colonna relativa al guasto.

#### 3.3.2 Bilanciamento del dataset

Prima di procedere con la fase di classificazione, ci siamo resi conto che il dataset era fortemente sbilanciato, ovvero il numero di righe relative ad un guasto del 10% erano molte di più rispetto alle righe relative a droni senza guasto, ovvero i dati risultanti sono stati:

- Nessun guasto: 390
- Guasto al 5%: 418
- Guasto al 10%: 541

Sulla base di questi risultati, si è valutato quindi di effettuare un oversampling del dataset. Per effettuare l'oversampling ci siamo basati sulla libreria **SMOTE** (Synthetic Minority Over-sampling Technique): viene quindi selezionata casualmente un'istanza della classe minoritaria, andando ad individuare i K punti più vicini ad essa. Andiamo poi a scegliere a caso uno di questi K punti ed andiamo a tracciare un segmento nello spazio delle feature tra il punto della classe minoritaria scelto ed il punto scelto tra i K più vicini. Le istanze sintetiche vengono generate come combinazione convessa di queste due istanze.

---

```

1 x = df_1.iloc[:, :-1]
2 y = df_1.iloc[:, -1:]
3
4 oversample = SMOTE()
5 x, y = oversample.fit_resample(x, y)

```

---

Codice 3.20: Oversampling del dataset

A questo punto, tutte e tre le categorie di guasto conterranno 541 elementi, potendo quindi procedere con il passo successivo di selezione delle feature più importanti.

### 3.3.3 Selezione delle feature

Per scegliere le feature migliori è risultato importante andare ad utilizzare un particolare metodo, in modo da poter prendere le feature più adatte su cui fare classificazione. Abbiamo deciso di utilizzare come tecnica statistica l'analisi della varianza (**ANOVA**). Con l'analisi della varianza l'obiettivo è appunto quello di individuare e quindi selezionare quelle feature più importanti al fine di ridurre la complessità del modello. L'idea di base è appunto individuare tutte quelle differenze tali per cui determinati valori siano relativi a caratteristiche proprie del gruppo di appartenenza.

---

```

1 fvalue_Best = SelectKBest(score_func=f_classif, k=15)
2 fit = fvalue_Best.fit(x, y)

```

---

Codice 3.21: Selezione delle 15 feature più importanti

Entrando nel dettaglio, abbiamo deciso di utilizzare l'analisi delle varianze in modo da poter individuare, sui dati a disposizione, le prime 15 feature più importanti.

---

```

1 features_score = pd.DataFrame(fit.scores_)
2 features = pd.DataFrame(x.columns)
3 feature_score = pd.concat([features, features_score], axis=1)
4 feature_score.columns = ["Input_Features", "F_Score"]
5 print(feature_score.nlargest(15, columns="F_Score"))

```

---

Codice 3.22: Score delle 15 feature individuate

Delle 15 feature individuate andiamo quindi a riportare gli score calcolati, Tabella 3.1.



Features	F_Score
freq1(AccX)	824.260891
freq1(GyrY)	821.015814
kurt(AccY)	802.798692
<>(AccY)	679.454279
amp1(AccX)	674.952150
amp1(C9)	634.326639
RMS(AccY)	608.729947
<>(AccX)	600.367193
<>(GyrY)	593.080986
amp2(AccX)	565.536257
amp2(C9)	490.150325
var(GyrY)	486.228107
RMS(GyrY)	467.758201
RMS(GyrZ)	466.862645
amp1(GyrY)	443.932238

Tabella 3.1: Score delle feature selezionate

A questo punto procediamo con la creazione di una lista che al suo interno conterrà le feature che non sono state selezionate dal metodo ANOVA, per andarle poi a rimuovere dall'insieme delle feature originali.

---

```

1 i=0
2 index_false=[]
3 for el in list(fvalue_Best.get_support()):
4     if not el:
5         index_false.append(i)
6     i=i+1
7
8 x=x.drop(x.columns[index_false],axis = 1)

```

---

Codice 3.23: Rimozione feature

Andiamo quindi ad utilizzare la lista creata per rimuovere dall'insieme delle feature originali, tutte le feature che non sono state scelte, in modo da poter tenere solo le feature rilevanti al fine della classificazione.

### 3.3.4 Separazione del dataset

Dopo aver individuato le feature più rilevanti, mediante il metodo ANOVA, procediamo con lo split del dataset in **training** e **test**, definendo una dimensione dell'insieme dei dati di training pari all'80% dell'insieme dei dati originali mentre per il test pari al 20%.

---

```

1 rs = 42 #random state
2
3 x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=rs)

```

---

Codice 3.24: Split del dataset training (80%) e test(20%)

In questo modo l'insieme di dati di training sarà composto da 1298 elementi mentre, l'insieme dei dati di test sarà composto da 325 elementi.

### 3.3.5 Definizione dei classificatori

Una volta effettuato lo split del dataset in training e test, procediamo con la definizione dei classificatori che andremo poi ad utilizzare.

---

```

1 # Lista dei classificatori:
2 classifiers = [
3     LogisticRegression(random_state = rs),
4     DecisionTreeClassifier(random_state=rs, criterion="entropy",
5         max_depth=1),
6     RandomForestClassifier(n_estimators = 1000,max_depth=1,
7         random_state=rs,bootstrap=False),
8     SVC(probability=True), #Support Vector Machine
9     MLPClassifier(random_state=rs), # Multi Layer Perceptron
10    SGDClassifier(random_state=rs) # Stochastic Gradient Descent
11 ]

```

---

Codice 3.25: Definizione dei classificatori

Per effettuare quindi la classificazione, abbiamo deciso di utilizzare sei tipologie di classificatori, in modo da poter analizzare i diversi approcci alla classificazione per ciascun modello ottenuto. Tra i classificatori utilizzati abbiamo:

- **LogisticRegression:** è un modello di regressione non lineare che viene utilizzato quando la variabile dipendente è di tipo dicotomico ma, può essere facilmente esteso utilizzando lo schema **ovr** (One-vs-Rest) ovvero suddividendo il problema di classificazione multiclasse in tanti classificatori binari quante sono le classi presenti nell'insieme dei dati.
- **Decision Tree:** è un algoritmo di apprendimento supervisionato non parametrico che si compone di una struttura ad albero gerarchica. L'apprendimento prevede l'utilizzo di una strategia "divide et impera".
- **Random Forest:** è un algoritmo di apprendimento automatico che combina l'output di più alberi decisionali per raggiungere un unico risultato

- **Support Vector Machine:** sono dei modelli di apprendimento supervisionato. Viene costruito un iperpiano (o insieme di iperpiani) in uno spazio a più dimensioni che viene quindi utilizzato per la classificazione.
- **Multi Layer Perceptron:** è un modello di rete neurale artificiale che mappa insiemi di dati in ingresso in un insieme di dati in uscita appropriati.
- **Stochastic Gradient Descent:** implementa dei modelli lineari regolarizzati con l'apprendimento della discesa stocastica del gradiente. Il gradiente della perdita viene stimato ad ogni campione e il modello viene aggiornato con learning rate.

Definiamo a questo punto tutte le liste in cui andremo a riportare tutti i nostri risultati che otterremo nella fase di classificazione.

---

```

1 clf_name = [] # nomi dei classificatori
2 model_results = pd.DataFrame.copy(y_test) #risultato della
  predizione dal modello
3
4 kfold = StratifiedKFold(n_splits=5) #cross-validation
5 cv_results = [] # scores della cross-validation
6 cv_acc = [] # accuratezza media della cross-validation, da
  massimizzare
7 cv_std = [] # deviazione standard della cross-validation, ma
  minimizzare
8
9 cnfm = [] #matrici di confusione
10 clr = [] #report della classificazione

```

---

Codice 3.26: Definizione delle liste che conterranno i risultati

La variabile *kfold* verrà utilizzata successivamente per effettuare la **cross-validation**, con un numero di split pari a 5. La **cross-validation** è una tecnica differente rispetto alla **hold-out**. Per quanto riguarda l'hold-out i dati di partenza vengono semplicemente suddivisi in dati di training e dati di test; facendo cio, non tutti i dati vengono utilizzati per addestrare il modello. Con la **K-fold Cross Validation** invece, l'insieme dei dati viene suddiviso in *k* parti di uguale numerosità ed, ad ogni passo, un sottoinsieme viene utilizzato per il test e i restanti *k-1* sottoinsiemi per il training (ad ogni passo il sottoinsieme dei dati di test viene cambiato con un altro).

### 3.3.6 Addestramento e validazione

Dopo aver definito tutte le specifiche utili al fine della classificazione, procediamo con l'addestramento e la validazione dei modelli ottenuti, con successiva valutazione dei risultati ottenuti.

Innanzitutto, definiamo il numero di classi su cui andremo a classificare i dati in quanto, la variabile *n\_classes* risulterà essere utile in fase di definizione delle curve ROC.

---

```

1 model = clf.fit(x_train, y_train.values.ravel())
2 y_pred = model.predict(x_test)
3 model_results[name] = y_pred

```

---

Codice 3.27: Addestramento e predizione

A questo punto, per ciascun classificatore in esame presente nella lista, procediamo con l'addestramento del modello, con successiva predizione con l'utilizzo dei dati di test.

---

```

1 cv_results.append(cross_val_score(clf, x, y, scoring = "accuracy",
    cv = kfold)) # cross validation
2 acc = round(accuracy_score(y_test.values.ravel(), y_pred), 2) #
    accuratezza semplice con training e test
3 print(f'Cross-Validation: {cross_val_score(clf, x, y, scoring = "
    accuracy",cv = kfold)}')
4
5 print(f'Accuracy: {acc} \t ---> {name} ')

```

---

Codice 3.28: Salvataggio dei risultati

Utilizzando i dati originali, calcoliamo i risultati della cross-validation con il classificatore che stiamo attualmente utilizzando ed inseriamo il risultato all'interno di una lista che conterrà il risultato ottenuto. Ci calcoliamo inoltre anche il valore dell'accuratezza calcolata con i risultati ottenuti in precedenza, data dalla percentuale di elementi correttamente classificati.

---

```

1 cnfm.append(confusion_matrix(y_test.values.ravel(), y_pred))
2 clr.append(classification_report(y_test.values.ravel(), y_pred))

```

---

Codice 3.29: Definizione delle matrici di confusione e report della classificazione

Con i risultati ottenuti in precedenza andiamo a raccogliere i dati per la costruzione delle matrici di confusione ed il report della classificazione.

---

```

1 if (j == 1) | (j == 2) | (j == 4):
2     y_score = clf.fit(x_train, y_train).predict_proba(x_test)
3 else:
4     clf = OneVsRestClassifier(clf)
5     y_score = clf.fit(x_train, y_train).decision_function(x_test)
6
7 lb = LabelBinarizer()
8 y_true_bin = lb.fit_transform(y_test)
9
10 fpr = dict()
11 tpr = dict()
12 roc_auc = dict()
13 for i in range(n_classes):
14     fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_score[:, i])
15     roc_auc[i] = auc(fpr[i], tpr[i])
16 colors = cycle(['blue', 'red', 'green'])
17 for i, color in zip(range(n_classes), colors):
18     plt.plot(fpr[i], tpr[i], color=color, lw=1.5,
19             label='ROC curve of class {0} (area = {1:0.2f})'

```

---

---

```

20         '%.format(i, roc_auc[i]))
21 plt.plot([0, 1], [0, 1], 'k--', lw=1.5)
22 plt.xlim([-0.05, 1.0])
23 plt.ylim([0.0, 1.05])
24 plt.xlabel('False Positive Rate')
25 plt.ylabel('True Positive Rate')
26 plt.title('Receiver operating characteristic for multi-class data'
27           )
28 plt.legend(loc="lower right")
29 plt.show()

```

---

Codice 3.30: Costruzione della curva ROC multiclasse

Successivamente, siamo andati a costruire la curva ROC, ovvero un grafico che mostra le performance del modello di classificazione considerando tutte le possibili soglie di classificazione. Per la realizzazione della curva ROC multiclasse abbiamo dovuto distinguere i classificatori in quanto alcuni di questi non supportano lo schema One-vs-Rest, è risultato quindi necessario calcolarci il valore dell *y\_score* con due metodologie differenti. Successivamente, si è proceduto poi a convertire le etichette multiclasse in etichette binarie, in modo quindi da avere un classificatore binario per ciascuna classe definita. Infine, si è proceduto con il plotting della curva ROC.

---

```

1 for i in cv_results:
2     cv_acc.append(i.mean())
3     cv_std.append(i.std())

```

---

Codice 3.31: Calcolo della media e della deviazione standard per la cross-validation

In questo passaggio procediamo, per ciascun classificatore, al calcolo della media e della deviazione standard per la valutazione dell'accuratezza calcolata con la cross-validation.

### 3.3.7 Matrici di confusione

Con i risultati precedentemente ottenuti si è proceduto con il plotting delle matrici di confusione.

---

```

1 plt.figure(figsize=(20,15))
2 sns.set(font_scale=1.4)
3 for i in range(len(classifiers)):
4     plt.subplot(3,3,i+1)
5     sns.heatmap(cnfm[i], annot=True, fmt="d", cmap="Reds")
6     plt.subplots_adjust(hspace = 0.5)
7     plt.xlabel('Predicted')
8     plt.ylabel('Actual')
9     plt.title(clf_name[i])

```

---

Codice 3.32: Plotting delle matrici di confusione

Le matrici di confusione vengono utilizzate per visualizzare il risultato ottenuto con la predizione della classe di appartenenza sui dati di test. È un metodo utile al fine

di visualizzare eventuali anomalie o semplicemente vedere la quantità di elementi che sono stati classificati correttamente. Nella diagonale principale sono riportati i dati che sono stati correttamente assegnati alla relativa classe di appartenenza, al di fuori della diagonale principale sono presenti tutti quei dati che non sono stati assegnati alla corretta classe di appartenenza.

### 3.3.8 Cross Validation Scores

Per analizzare meglio i risultati ottenuti con la cross-validation, siamo andati ad utilizzare un grafico per l'analisi dell'accuratezza media ed errore risultante dalla cross-validation.

---

```

1 cv_res = pd.DataFrame({"CrossValMeans":cv_acc, "CrossValerrors":
    cv_std,"Algorithm":clf_name})
2
3 plt.figure(figsize=(12,6))
4 sns.barplot(x="CrossValMeans",y="Algorithm",data = cv_res, palette
    ="Set2",orient = "h",**{'xerr':cv_std})
5 plt.xlabel("Mean Accuracy")
6 plt.title("Cross validation scores")

```

---

Codice 3.33: Plotting Cross-Validation Scores

In questo modo possiamo visualizzare l'accuratezza media di ciascun classificatore (mediante un grafico a barre) ed il relativo errore, calcolato mediante deviazione standard.

### 3.3.9 Report Classificazione

Ulteriori valori che siamo andati a calcolare sono quelli relativi ad un report sulla classificazione.

---

```

1 for i in range(len(classifiers)):
2     print (f"{clf_name[i]} Classification Report:" )
3     print (clr[i])

```

---

Codice 3.34: Report classificazione

Per ciascun classificatore, ci siamo andati a calcolare:

- **precision**: è data dal numero di elementi correttamente classificati di quella classe rispetto al numero di elementi predetti di quella classe
- **recall**: è data dal numero di elementi correttamente classificati di quella classe rispetto al numero di elementi veri della classe
- **f1-score**: è una combinazione di precision e recall
- **supporto**: è il numero di dati di test utilizzati nella predizione

- **accuratezza:** è l'accuratezza che si ottiene dalla divisione del dataset in training e test, andando a vedere quindi quanti elementi di quella classe sono stati classificati correttamente, rispetto al numero totale di elementi di quella classe
- **macro avg:** è il calcolo della media riportando il problema della classificazione ad una classificazione binaria
- **weighted avg:** è la media pesata, ovvero ciascun elemento viene moltiplicato per un peso durante il calcolo della media

# Capitolo 4

## Risultati

Avendo riportato tutti i passaggi relativi allo sviluppo del nostro progetto, procediamo con il mostrare i risultati ottenuti, dall'addestramento dei diversi classificatori utilizzati. Procederemo con riportare il risultato dell'accuratezza semplice, calcolata dividendo il dataset in training e test, successivamente procederemo con i risultati dell'accuratezza ottenuti mediante cross-validation, con un numero di fold pari a 5. Per ciascun classificatore verrà riportata anche la relativa curva ROC, mostrando la curva per ciascuna classe predetta. Riporteremo quindi i risultati numerici, rappresentati mediante matrici di confusione, relativi alla predizione ottenuta applicando i dati di test al modello e report sulla classificazione. Verrà riportato infine un grafico a barre rappresentante gli score della cross-validation, con accuratezza media e deviazione standard.

### 4.1 Logistic Regression

Con il Logistic Regression riusciamo ad ottenere un'accuratezza pari all'88%. Il modello ottenuto è in grado di effettuare una separazione distinta delle classi, soprattutto è capace nel distinguere un volo relativo ad un drone senza guasto da un volo con guasto al 10%. Maggiore difficoltà viene riscontrata nella distinzione tra un volo relativo ad un drone senza guasto da un drone con guasto al 5%. Di seguito riportiamo i risultati delle accuratezze ottenute mediante cross-validation:

0.89230769   0.88923077   0.84923077   0.93518519   0.87037037

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratezze che otteniamo sono accettabili e relativamente alte, ma anche conformi al risultato ottenuto dal calcolo dell'accuratezza semplice.



### 4.1.1 Curva ROC

Per quanto riguarda il Logistic Regression, la curva ROC ottenuta è riportata in figura 4.1. Come possiamo vedere la curva migliore è quella relativa alla classe 2

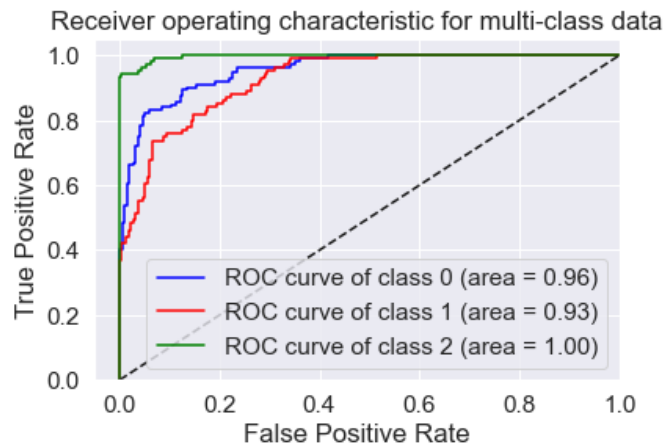


Figura 4.1: Curva ROC Logistic Regression

(ovvero riferito a volo con guasto al 10%) che abbraccia perfettamente l'angolo in alto a sinistra del grafico riportato, con AUC pari a 1. Anche la curva relativa alla classe 0 (nessun guasto) è buona con un'AUC pari a 0.96. Più bassa è invece la curva, e rispettivo AUC, relativamente al volo con guasto al 5%.

### 4.1.2 Matrice di confusione

Riportiamo in figura 4.2 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

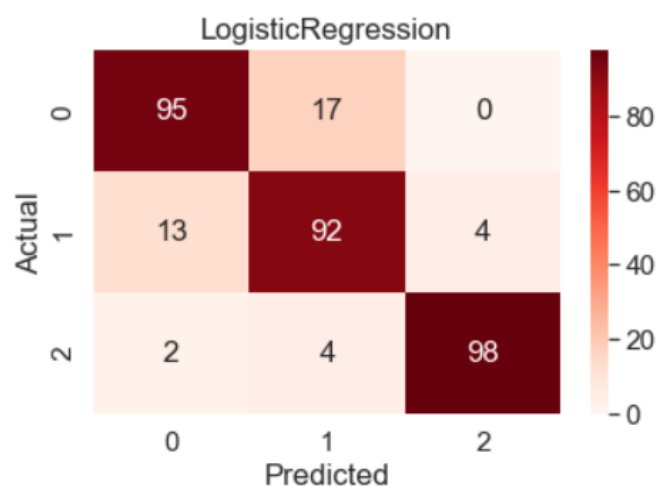


Figura 4.2: Matrice di confusione Logistic Regression

Come possiamo vedere, il modello è in grado di effettuare una netta distinzione tra dati appartenenti alla classe 0 (nessun guasto) e dati appartenenti alla classe 2 (guasto al 10%). Maggiore difficoltà è riscontrata nella distinzione tra i dati appartenenti alla classe 1 (guasto al 5%) ed i dati appartenenti alla classe 0 (nessun guasto).

### 4.1.3 Report della classificazione

Riportiamo inoltre in tabella 4.1 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.86	0.85	0.86	112
classe 1	0.81	0.84	0.83	109
classe 2	0.96	0.94	0.95	104
accuracy			0.88	325
macro avg	0.88	0.88	0.88	325
weighted avg	0.88	0.88	0.88	325

Tabella 4.1: Report Classificazione Logistic Regression

Come possiamo vedere, per quanto riguarda la classe 2, risultano dei valori più alti di precision, recall e F1-score, rispetto alle altre classi, in quanto commettiamo un minor numero di errori.

## 4.2 Decision Tree

Con il Decision Tree riusciamo ad ottenere un'accuratezza pari al 65%. In questo caso l'accuratezza classica risulta essere di molto più bassa rispetto al classificatore precedente. Il modello è in grado di distinguere un volo con drone senza guasto da un volo con drone con guasto al 10% ma, trova estrema difficoltà nell'individuare i voli relativi a droni con guasto al 5%. Di seguito riportiamo i risultati delle accuratèzze ottenute mediante cross-validation:

0.66153846   0.65846154   0.65538462   0.65123457   0.66049383

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratèzze che otteniamo sono relativamente basse, ma comunque conformi al risultato ottenuto dal calcolo dell'accuratezza semplice.

### 4.2.1 Curva ROC

Per quanto riguarda il Decision Tree, la curva ROC ottenuta è riportata in figura 4.3.

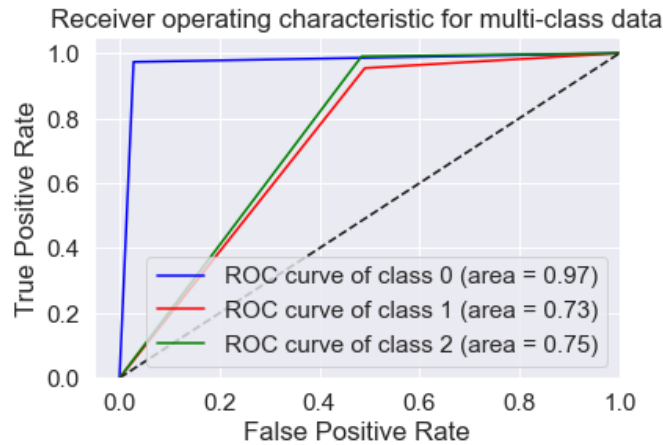


Figura 4.3: Curva ROC Decision Tree

In questo caso, la curva ROC relativa alla classe 0 (volo senza guasto) abbraccia molto bene l'angolo in alto a sinistra, rispetto invece alle altre due curve, in quanto il classificatore non è in grado di distinguere i voli di classe 1 dai voli di classe 2. Il valore AUC è molto buono per la classe 0, risulta essere invece più basso per le altre due classi.

### 4.2.2 Matrice di confusione

Riportiamo in figura 4.4 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

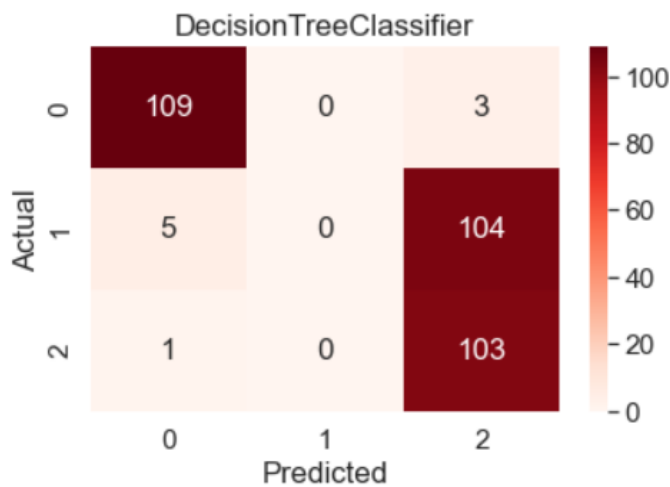


Figura 4.4: Matrice di confusione Decision Tree

Come possiamo osservare, il modello è in grado di individuare gli elementi di classe 0 correttamente ma, ha estrema difficoltà nell'andare a riconoscere gli elementi di classe 1, andandoli a classificare tutti di classe 2, comportando quindi la generazione di falsi allarmi in caso di guasto al 5% che viene segnalato come guasto al 10%.

### 4.2.3 Report della classificazione

Riportiamo inoltre in tabella 4.2 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.95	0.97	0.96	112
classe 1	0.00	0.00	0.00	109
classe 2	0.49	0.99	0.66	104
accuracy			0.65	325
macro avg	0.48	0.65	0.54	325
weighted avg	0.48	0.65	0.54	325

Tabella 4.2: Report Classificazione Decision Tree

Come possiamo osservare, i valori più alti li otteniamo per la classe 0, in quanto il modello è in grado di distinguere il volo di un drone senza guasto dalle altre casistiche. Diversa è la situazione per i valori relativi alla classe 1, in quanto nessun dato è stato classificato correttamente. Discreti invece sono i risultati relativi alla classe 2, con una recall pari allo 0.99 in quanto solo un elemento di classe 2 è stato classificato di classe 0 (come possiamo appunto osservare dalle matrici di confusione). Diverso è invece il valore della precision (0.49) in quanto sono stati predetti di classe 2 anche gli elementi di classe 1.

## 4.3 Random Forest

Con il Random Forest riusciamo ad ottenere un'accuratezza pari al 95%. In questo caso l'accuratezza classica risulta essere la più alta che riusciamo ad ottenere, rispetto alle accuratze di tutti gli altri classificatori. È in grado di distinguere tutte le diverse categorie di classificazione in modo corretto, senza compiere errori rilevanti. Distigue perfettamente i voli relativi a droni senza guasto da voli relativi a droni con guasto al 10%. Presenta una lieve difficoltà nel distinguere voli di droni con guasto al 5% da voli di droni con guasto al 10% ed anche con voli di droni

senza guasto. Di seguito riportiamo i risultati delle accuratezze ottenute mediante cross-validation:

0.93538462   0.96615385   0.94461538   0.94753086   0.94753086

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratezze che otteniamo sono relativamente alte, ma comunque conformi al risultato ottenuto dal calcolo dell'accuratezza semplice.

### 4.3.1 Curva ROC

Per quanto riguarda il Random Forest, la curva ROC ottenuta è riportata in figura 4.5.

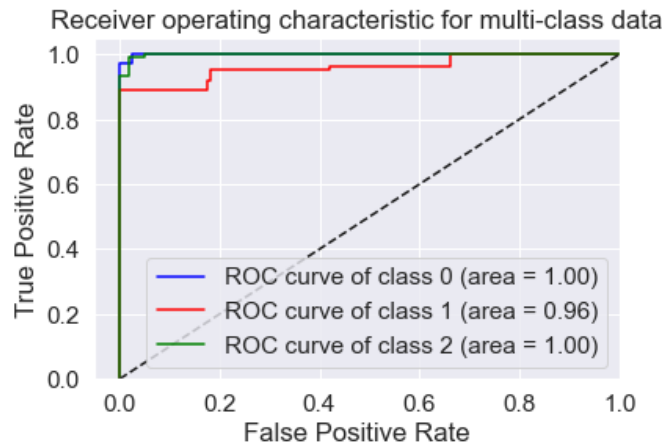


Figura 4.5: Curva ROC Random Forest

Come possiamo osservare, la classe 0 e la classe 2 hanno entrambe una curva ROC che abbraccia perfettamente l'angolo in alto a sinistra del grafico, significato del fatto che entrambe sono perfettamente distinguibili, con un valore dell'AUC pari a 1 per entrambe le classi. Simile è la situazione la classe 1 con una curva ROC leggermente diversa e più spostata verso il basso in alcuni punti ed un valore dell'AUC pari a 0.96.

### 4.3.2 Matrice di confusione

Riportiamo in figura 4.6 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

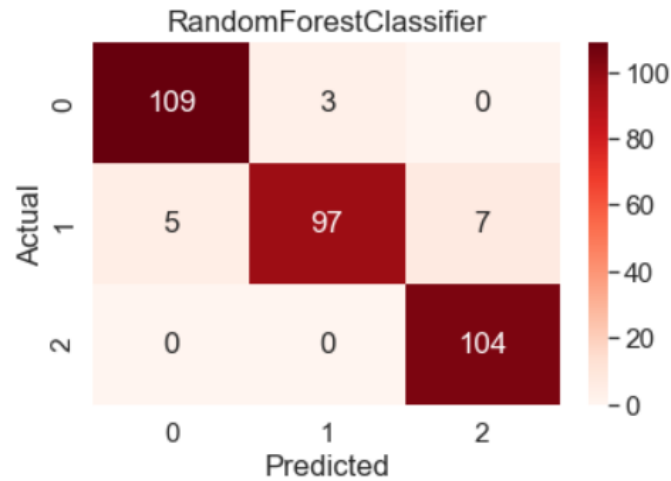


Figura 4.6: Matrice di confusione Random Forest

Come possiamo osservare, quasi tutti i dati presenti nell'insieme di test sono stati assegnati alla classe corretta. Si può osservare una lieve sfumatura nell'assegnazione degli elementi di classe 1 alle altre classi mentre, tutti gli elementi di classe 2 vengono correttamente assegnati alla rispettiva classe di appartenenza, senza commettere alcun errore. Nel complesso il risultato che otteniamo è molto buono.

### 4.3.3 Report della classificazione

Riportiamo inoltre in tabella 4.3 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.96	0.97	0.96	112
classe 1	0.97	0.89	0.93	109
classe 2	0.94	1.00	0.97	104
accuracy			0.95	325
macro avg	0.95	0.95	0.95	325
weighted avg	0.95	0.95	0.95	325

Tabella 4.3: Report Classificazione Random Forest

Per quanto riguarda il Random Forest, otteniamo risultati molto alti rispetto a classificatori precedenti, soprattutto per quanto riguarda la recall della classe 2,

avendo classificato correttamente tutti gli elementi ad essa appartenenti. Valori più bassi li otteniamo nella classe 1 per quanto riguarda i valori di Recall ed F1-score.

## 4.4 Support Vector Machine

Con il Support Vector Machine riusciamo ad ottenere un'accuratezza pari al 67%. In questo caso l'accuratezza classica risulta essere di molto più bassa rispetto al classificatore precedente. In questo caso il modello risultante non presenta grandi difficoltà nella classificazione di voli di drone con guasto al 10%. Non possiamo però dire la stessa cosa però per quanto riguarda la distinzione tra i voli relativi a droni senza guasto rispetto a voli relativi a droni con guasto al 5%, in quanto il classificatore presenta delle difficoltà nell'assegnazione degli elementi alle rispettive classi. Di seguito riportiamo i risultati delle accuratèzze ottenute mediante cross-validation:

0.68307692   0.67384615   0.63076923   0.69753086   0.65740741

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratèzze che otteniamo sono relativamente basse, ma relativamente conformi al risultato ottenuto dal calcolo dell'accuratezza semplice.

### 4.4.1 Curva ROC

Per quanto riguarda il Support Vector Machine, la curva ROC ottenuta è riportata in figura 4.7.

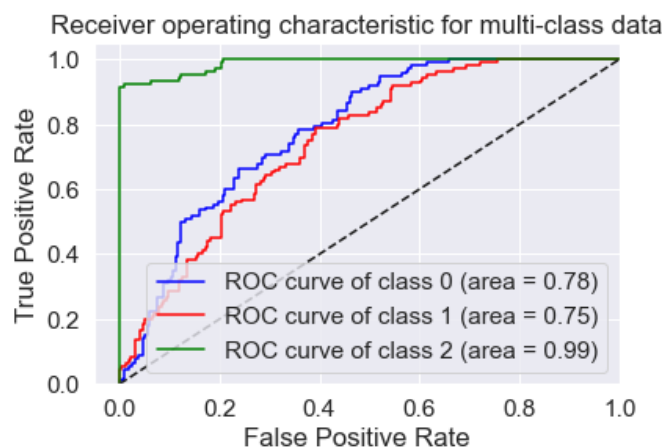


Figura 4.7: Curva ROC Support Vector Machine

Come possiamo osservare dal grafico riportato, la curva ROC relativa alla classe 2 abbraccia l'angolo in alto a sinistra, a ragion del fatto che il classificatore è in grado di distinguere correttamente gli elementi classe 2 dagli altri elementi appartenenti alle altre classi, con un valore AUC pari allo 0.99. Le altre due classi hanno una

curva ROC decisamente più bassa rispetto alla precedente, relative al fatto che il classificatore fa difficoltà nella distinzione dei dati. La curva ROC della classe 0 è leggermente migliore della curva ROC della classe 1 con un AUC di 0.75 mentre di 0.78 per la classe 0.

#### 4.4.2 Matrice di confusione

Riportiamo in figura 4.8 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

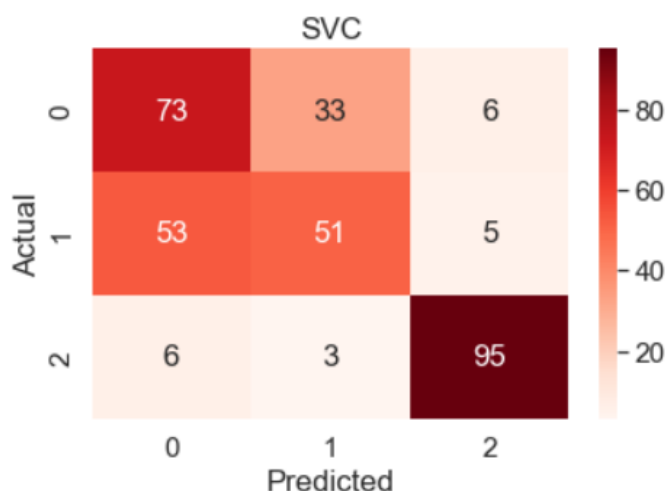


Figura 4.8: Matrice di confusione Random Forest

Come possiamo osservare, non vi sono errori rilevanti per quanto riguarda l'assegnazione degli elementi di classe 2. Particolari differenze invece le possiamo osservare nell'assegnazione degli elementi di classe 0 e di classe 1. Per quanto riguarda la classe dei voli relativi a droni senza guasto (classe 0), il classificatore è in grado di fare una corretta distinzione tra questi e i voli relativi a droni con guasto al 10% (guasto 2), presenta invece una maggiore difficoltà nel distinguere voli senza guasto da voli con guasto al 5%. Comunque, in questa situazione, la maggior parte dei dati di classe 0 è assegnato correttamente (su 112 elementi di classe 0, 73 sono assegnati alla classe corretta, 33 alla classe 1 e 6 alla classe 2). Un discorso simile può essere fatto per gli elementi di classe 1: in questo caso però, poco meno della metà degli elementi è assegnato alla classe corretta (su 109 elementi di classe 1, 51 sono assegnati alla classe corretta, 53 sono assegnati alla classe 0 e 5 alla classe 2). Questo classificatore commette quindi un errore maggiore nell'assegnazione dei dati di test alle relative classi di appartenenza.



### 4.4.3 Report della classificazione

Riportiamo inoltre in tabella 4.4 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.55	0.65	0.60	112
classe 1	0.59	0.47	0.52	109
classe 2	0.90	0.91	0.90	104
accuracy			0.67	325
macro avg	0.68	0.68	0.67	325
weighted avg	0.67	0.67	0.67	325

Tabella 4.4: Report Classificazione Support Vector Machine

Come possiamo osservare dai risultati presenti nel report, questi rispecchiano le affermazioni fatte in precedenza. I valori più alti di precision, recall ed F1-score li otteniamo per la classe 2, non avendo particolari problemi nell'andare ad assegnare i dati relativi a tale classe. Valori più bassi li otteniamo invece per la classe 1: come avevamo osservato nelle matrici di confusione, sono stati correttamente classificati meno della metà dei dati relativi a tale classe, motivo per cui abbiamo un valore di recall pari allo 0.47.

## 4.5 Multi Layer Perceptron

Con il Multi Layer Perceptron riusciamo ad ottenere un'accuratezza pari al 91%. In questo caso il risultato dell'accuratezza è relativamente alto, in questo caso il classificatore è in grado di distinguere abbastanza bene gli elementi delle diverse classi, con qualche incertezza, soprattutto nell'assegnazione degli elementi di classe 1. Di seguito riportiamo i risultati delle accuratèzze ottenute mediante cross-validation:

0.88   0.93846154   0.89230769   0.73148148   0.91358025

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratèzze che otteniamo sono relativamente alte. In questa casistica osserviamo però una maggiore variazione dell'accuratezza rispetto ai classificatori precedenti, soprattutto perché, per quanto riguarda la quarta iterazione, otteniamo un'accuratezza di circa 0.73, rispetto alle accuratèzze degli altri fold che sono più alte ed anche più conformi all'accuratezza classica.

### 4.5.1 Curva ROC

Per quanto riguarda il Multi Layer Perceptron, la curva ROC ottenuta è riportata in figura 4.9.

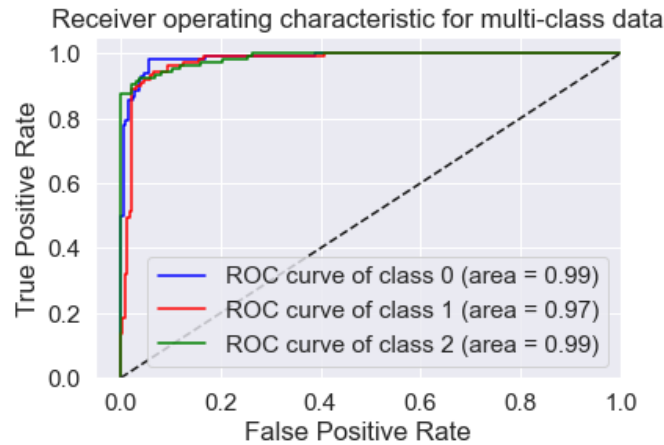


Figura 4.9: Curva ROC Multi Layer Perceptron

Come possiamo osservare dal grafico riportato, tutte e tre le curve ROC presentano un andamento simile ed abbracciano molto bene l'angolo in alto a sinistra. Sia la classe 0 che la classe 2 hanno un valore AUC pari a 0.99, l'AUC relativo alla classe 1 è invece un po' più basso, pari allo 0.97, sempre a causa del fatto che si fa maggiore difficoltà nell'assegnare i dati a questa classe.

### 4.5.2 Matrice di confusione

Riportiamo in figura 4.10 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

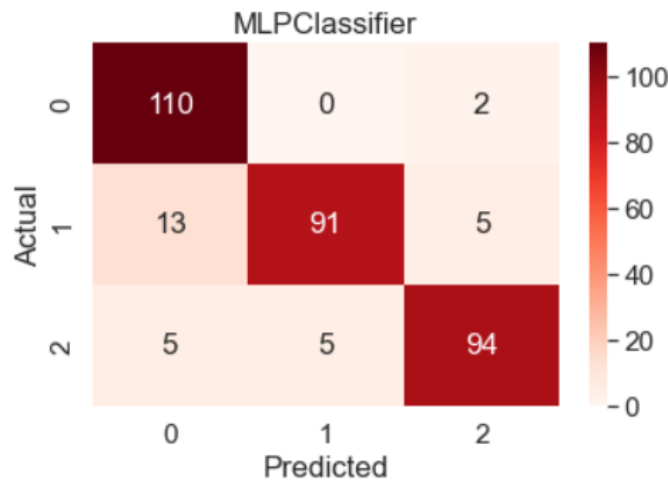


Figura 4.10: Matrice di confusione Multi Layer Perceptron

Come possiamo vedere, i dati che vengono classificati in modo migliore sono quelli relativi alla classe 0, in cui solo 2 elementi vengono assegnati ad una classe diversa da quella corretta. Maggiore differenza la riscontriamo sempre nell'assegnazione degli elementi di classe 1, in cui una discreta quantità viene assegnata erroneamente alla classe 0 ed una piccola quantità alla classe 2; tutto sommato la maggior parte degli elementi viene assegnato alla classe corretta. Simili affermazioni possiamo farle per la classe 2.

### 4.5.3 Report della classificazione

Riportiamo inoltre in tabella 4.6 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.86	0.98	0.92	112
classe 1	0.95	0.83	0.89	109
classe 2	0.93	0.90	0.92	104
accuracy			0.91	325
macro avg	0.91	0.91	0.91	325
weighted avg	0.91	0.91	0.91	325

Tabella 4.5: Report Classificazione Multi Layer Perceptron

Come possiamo osservare dai risultati ottenuti, il risultato più basso di precision lo otteniamo nella classe 0, in quanto alcuni elementi di classe 1 vengono predetti come di classe 0. Di conseguenza, il valore più basso di recall lo osserviamo nella classe 1 in quanto non tutti gli elementi sono stati classificati correttamente.

## 4.6 Stochastic Gradient Descent

Con il Stochastic Gradient Descent riusciamo ad ottenere un'accuratezza pari al 63%. In questo caso il risultato dell'accuratezza è relativamente basso, in quanto il classificatore è in grado di distinguere abbastanza bene gli elementi della classe 0 (volo relativo a drone senza guasto) dagli elementi della classe 2 (volo relativo a drone con guasto al 10%) ma, trova estrema difficoltà nel riconoscere gli elementi di classe 1 (volo relativo a drone con guasto al 5%). Di seguito riportiamo i risultati delle accuratèzze ottenute mediante cross-validation:

0.67384615   0.66153846   0.75076923   0.74382716   0.62037037

Come possiamo vedere dai risultati ottenuti, per ciascun test fatto sui fold, le accuratezze che otteniamo sono un po' più alte rispetto all'accuratezza semplice calcolata in precedenza. In questa casistica osserviamo inoltre una maggiore variazione dell'accuratezza in ciascun fold ed, in particolare, l'ultimo fold presenta un'accuratezza più bassa rispetto alle altre calcolate nei fold precedenti, ma comunque conforme con l'accuratezza classica.

#### 4.6.1 Curva ROC

Per quanto riguarda il Stochastic Gradient Descent, la curva ROC ottenuta è riportata in figura 4.11.

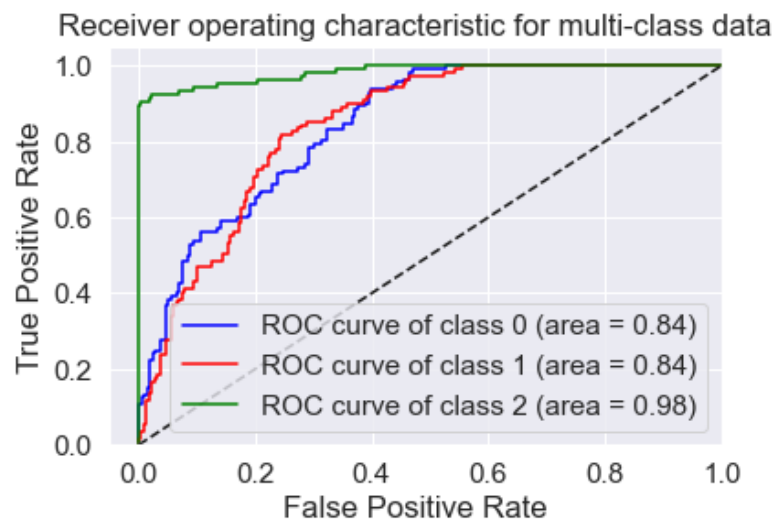


Figura 4.11: Curva ROC Stochastic Gradient Descent

Come possiamo osservare nell'immagine riportata, la curva ROC relativa alla classe 2 abbraccia molto bene l'angolo in alto a sinistra, con un valore dell'AUC pari a 0.98; questo significa che il modello è in grado di distinguere bene gli elementi di classe 2 dalle altre casistiche. Diversa è invece la situazione della classe 0 e della classe 1, con entrambe un valore dell'AUC pari a 0.84: questo significa che il classificatore non è in grado di distinguere correttamente gli elementi della classe 0 da quelli della classe 1.

### 4.6.2 Matrice di confusione

Riportiamo in figura 4.12 i risultati numerici della predizione mediante matrice di confusione, applicando l'insieme dei dati di test al modello addestrato.

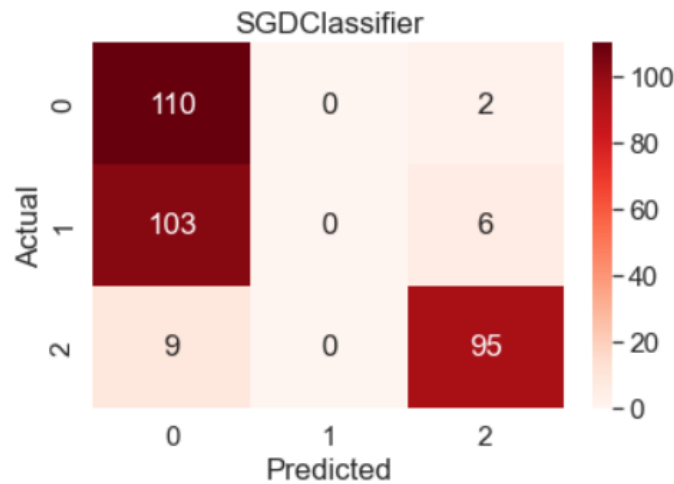


Figura 4.12: Matrice di confusione Stochastic Gradient Descent

Come possiamo vedere nella matrice di confusione, il modello è in grado di distinguere abbastanza bene gli elementi di classe 2 dalle altre casistiche, con qualche incertezza. Lo stesso però non possiamo dirlo per gli elementi di classe 0 e 1: gli elementi classe 0 vengono comunque correttamente assegnati alla rispettiva classe ma, anche gli elementi di classe 1 vengono assegnati alla classe 0. Da un lato questo potrebbe essere un male perché molti guasti relativi alla classe 1 non verrebbero segnalati, dall'altro lato però l'allarme viene generato solamente in casi reali e critici, ovvero quando si verifica un guasto del 10%. Seppur quindi sbagliando, questo classificatore risulta essere migliore del modello ottenuto con il Decision Tree, che andava a classificare gli elementi di classe 1 come di classe 2, generando molti falsi allarmi. In questo caso quindi l'allarme verrebbe generato solamente in casi di reale guasto critico.

### 4.6.3 Report della classificazione

Riportiamo inoltre in tabella 4.6 i risultati ottenuti relativamente alla precision, recall, f1-score, supporto, accuracy, macro avg, weighted avg.

	Precision	Recall	F1-score	Supporto
classe 0	0.50	0.98	0.66	112
classe 1	0.00	0.00	0.00	109
classe 2	0.92	0.91	0.92	104
accuracy			0.63	325
macro avg	0.47	0.63	0.53	325
weighted avg	0.47	0.63	0.52	325

Tabella 4.6: Report Classificazione Stochastic Gradient Descent

Come possiamo osservare dai valori riportati nel report, i valori migliori li otteniamo per la classe 2, a conferma del fatto che il classificatore è in grado di distinguere bene gli elementi appartenenti a tale classe. I valori relativi alla classe 1 invece sono tutti pari a 0, in quanto nessun dato viene classificato correttamente. Per quanto riguarda la classe 0 invece, abbiamo un valore di recall pari a 0.98 a conferma del fatto che gli elementi di tale classe vengono classificati correttamente mentre, si ottiene un valore di precision pari a 0.50 in quanto quasi tutti gli elementi di classe 1 vengono classificati come di classe 0.

## 4.7 Cross-Validation Scores

Nell'immagine 4.13 riportiamo gli score relativi alla cross-validation riportante l'accuratezza media e la deviazione standard per ciascun classificatore utilizzato.

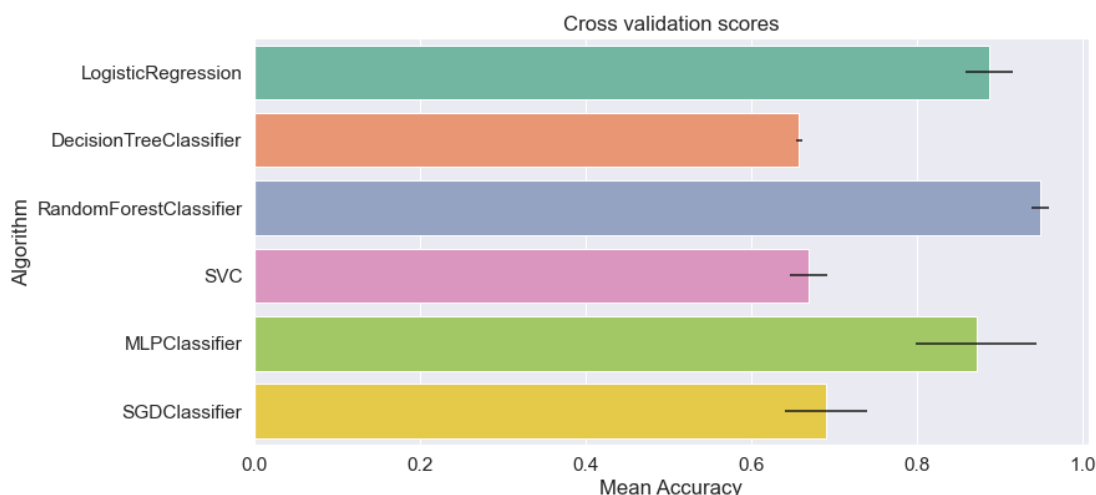


Figura 4.13: Cross-Validation Scores

Come possiamo osservare dal grafico a barre, viene riportata l'accuratezza media con relativo errore, riguardante la variazione delle accuratèzze. L'errore minore è presente nel Decision Tree e nel Random Forest, rappresentante il fatto che l'accuratezza dei vari fold non si discosta dal valore dell'accuratezza media riportata. Il Logistic Regression e il Support Vector Machine hanno comunque un errore del tutto accettabile. Un discorso diverso viene invece fatto per il Multi Layer Perceptron ed il Stochastic Gradient Descent, aventi entrambi un errore abbastanza significativo, conseguenza del fatto che hanno un'accuratezza del tutto variabile per ciascun fold. I risultati migliori li otteniamo per il Random Forest, sia in termini di accuratezza migliore che in termini di variabilità delle accuratèzze in ciascun fold.

## Capitolo 5

### Conclusioni e Sviluppi futuri

Con lo sviluppo di questo progetto si è proceduto alla realizzazione di modelli in grado di predire la classe di guasto relativa al drone in analisi. Abbiamo prima convertito il dataset iniziale fornitoci in formato .mat, in tanti file .csv, ognuno di essi contenente le matrici relative alle variabili utili al fine del calcolo di feature rilevanti.

Sono stati poi estratti in Python i dati dai file csv, effettuato un trimming del dataset relativo a ciascuna variabile individuata e sincronizzati i tempi con una frequenza pari a 350 Hz, in modo che in ogni misurazione si distanziasse dall'altra di 2857  $\mu$ s. A questo punto si è proceduto con il calcolo delle feature nel tempo ed in frequenza, su una finestra temporale di un secondo. I risultati sono stati poi memorizzati su un file .csv, uno per ciascun volo analizzato, per un totale di 18 file csv.

Avendo ottenuto le feature, abbiamo creato un unico dataframe e rimosso la colonna dei tempi, non utile ai fini delle analisi. Abbiamo poi effettuato un oversampling al fine di incrementare la quantità dei dati e selezionato le 15 feature più importanti mediante metodo ANOVA.

Dopo tali accorgimenti, si è proceduto a dividere il dataset in training e test, ai fini della classificazione. Siamo quindi andati ad addestrare diversi classificatori: Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, Multi Layer Perceptron e Stochastic Gradient Descent. Qui abbiamo quindi calcolato l'accuratezza semplice, la curva ROC, la matrice di confusione ed effettuato un report sulla classificazione. È stata inoltre utilizzata la cross-validation con 5 fold, al fine di evidenziare eventuali differenze con l'hold out e mostrare un'accuratezza media con relativa deviazione standard mediante grafico a barre. Dalle analisi effettuate, il classificatore migliore è risultato il Random Forest, in grado di distinguere correttamente tutte le classi definite.



## Sviluppi futuri

Tale progetto pone le basi per il miglioramento e lo sviluppo di utilità che possano facilitare l'utilizzo dei modelli di classificazione sviluppati.

Vi sono ancora diversi elementi che possono essere introdotti al fine del perfezionamento dei modelli, come, ad esempio, la **Hyperparameter Optimization**. La **Hyperparameter Optimization** consiste nella scelta dei parametri ottimali per un modello di Machine Learning, al fine di migliorarne le performances. Ci sono diverse tecniche per ottimizzare la scelta degli iperparametri, tra queste abbiamo la **Grid Search**: tale tecnica consiste nell'andare a prendere un insieme dei valori per i parametri che si vogliono ottimizzare e provare tutte le possibili combinazioni, andando quindi a prendere il modello migliore risultante.

Si può pensare ulteriormente ad effettuare un **Model Ensembling** mediante **Voting Classifier**: vengono combinati algoritmi di Machine Learning concettualmente differenti al fine di prevedere la classe di appartenenza dei dati passati al modello risultante dalla combinazione. Sul modello risultante dal Voting Classifier si può quindi pensare di estrarre la curva ROC, la curva Precision Recall e la Learning Curve.

## Conclusioni

Alla luce di quanto detto e visto fin'ora risulta evidente come le possibilità di sviluppo e ampliamento di questo progetto siano molte, soprattutto per la realizzazione di un modello derivante dall'ensembling di più classificatori. L'esperienza di sviluppo di questo progetto si è rivelata altamente formativa e ci ha permesso di approfondire le conoscenze riguardo il rilevamento di un guasto a partire da log di volo.

Il lavoro svolto rappresenta un importante punto di partenza e una base solida per lo sviluppo di altri progetti, inglobando altre funzionalità e ottimizzazioni prima di poter essere utilizzato su larga scala.

# Bibliografia

- [1] *ArduPilot: Downloading and Analyzing Data Logs in Mission Planner*. URL: <https://ardupilot.org/copter/docs/common-downloading-and-analyzing-data-logs-in-mission-planner.html>.
- [2] *Python 3.10 documentation*. URL: <https://docs.python.org/3.10/>.
- [3] *MATLAB*. URL: <https://it.mathworks.com/products/matlab.html>.
- [4] *Jupyter Notebook*. URL: <https://jupyter.org/>.

# Elenco delle figure

3.1	Lista delle cartelle che contengono il relativo file .mat . . . . .	7
3.2	Rappresentazione matriciale ed intestazione relativa alla variabile ATT . . . . .	8
3.3	Workspace contenente tutte le matrici e relative intestazioni . . . .	9
3.4	Elenco file generati . . . . .	9
3.5	File csv generato . . . . .	10
4.1	Curva ROC Logistic Regression . . . . .	29
4.2	Matrice di confusione Logistic Regression . . . . .	29
4.3	Curva ROC Decision Tree . . . . .	31
4.4	Matrice di confusione Decision Tree . . . . .	31
4.5	Curva ROC Random Forest . . . . .	33
4.6	Matrice di confusione Random Forest . . . . .	34
4.7	Curva ROC Support Vector Machine . . . . .	35
4.8	Matrice di confusione Random Forest . . . . .	36
4.9	Curva ROC Multi Layer Perceptron . . . . .	38
4.10	Matrice di confusione Multi Layer Perceptron . . . . .	38
4.11	Curva ROC Stochastic Gradient Descent . . . . .	40
4.12	Matrice di confusione Stochastic Gradient Descent . . . . .	41
4.13	Cross-Validation Scores . . . . .	43

# Elenco delle tabelle

1.1	Informazioni di attitudine . . . . .	2
1.2	Feedback ricevuto dagli ESC . . . . .	2
1.3	Informazioni relative all'accelerometro e giroscopio . . . . .	3
1.4	Valori di uscita servo channel da 9 a 14 . . . . .	3
1.5	Informazioni di attitudine . . . . .	3
3.1	Score delle feature selezionate . . . . .	21
4.1	Report Classificazione Logistic Regression . . . . .	30
4.2	Report Classificazione Decision Tree . . . . .	32
4.3	Report Classificazione Random Forest . . . . .	34
4.4	Report Classificazione Support Vector Machine . . . . .	37
4.5	Report Classificazione Multi Layer Perceptron . . . . .	39
4.6	Report Classificazione Stochastic Gradient Descent . . . . .	42

# Elenco dei codici

3.1	Script per convertire in file .csv le matrici presenti nei file .mat . . .	8
3.2	Estrazione dati RCOU . . . . .	11
3.3	Preso tempo di inizio e tempo di fine del volo . . . . .	11
3.4	Estrazione dati ATT e XKF . . . . .	11
3.5	Confronto valori ATT e XFK . . . . .	12
3.6	Estrazione dati IMU . . . . .	13
3.7	Unione delle tabelle . . . . .	14
3.8	Settaggio istante di partenza pari a 0 . . . . .	14
3.9	Sincronizzazione con 350 misurazioni al secondo . . . . .	14
3.10	Formattazione del tempo . . . . .	15
3.11	Definita funzione per il calcolo del Root Mean Square . . . . .	15
3.12	Definizione funzione per il calcolo delle feature nel tempo e in frequenza . . . . .	16
3.13	Calcolo feature per ogni variabile . . . . .	17
3.14	Creazione del dataset . . . . .	17
3.15	Salvataggio in .csv del dataset . . . . .	17
3.16	Creazione del Dataframe . . . . .	18
3.17	Rimozione di eventuali duplicati . . . . .	19
3.18	Shuffling del dataframe . . . . .	19
3.19	Rimozione della colonna "tempo" . . . . .	19
3.20	Oversampling del dataset . . . . .	20
3.21	Selezione delle 15 feature più importanti . . . . .	20
3.22	Score delle 15 feature individuate . . . . .	20
3.23	Rimozione feature . . . . .	21
3.24	Split del dataset training (80%) e test(20%) . . . . .	22
3.25	Definizione dei classificatori . . . . .	22
3.26	Definizione delle liste che conterranno i risultati . . . . .	23
3.27	Addestramento e predizione . . . . .	24
3.28	Salvataggio dei risultati . . . . .	24
3.29	Definizione delle matrici di confusione e report della classificazione .	24
3.30	Costruzione della curva ROC multiclasse . . . . .	24
3.31	Calcolo della media e della deviazione standard per la cross-validation	25
3.32	Plotting delle matrici di confusione . . . . .	25
3.33	Plotting Cross-Validation Scores . . . . .	26
3.34	Report classificazione . . . . .	26