Relazione Progetto Machine Learning e Data Mining

Bernocchi Alessandro, Cherubini Mattia, Metelli Claudio

3 gennaio 2025

Indice

1 Introduzione

L'obiettivo di questo lavoro è quello di realizzare una classificazione binaria nell'ambito dell'attivazione di un allarme antincendio, servendoci dei dati sulla rilevazione del fumo nell'aria raccolti da diversi sensori IoT, presenti nel dataset **Smoke Detection**.
Lo studio si basa su fenomeni fisici misurabili, come la concentrazione di particelle nell'aria, la temperatura, l'umidità, la pressione e altre grandezze significative, che producono dati numerici e strutturati.
Il processo di analisi e sviluppo seguirà le seguenti fasi principali:

- Caricamento e analisi preliminare del dataset, per comprendere la struttura dei dati e le loro caratteristiche principali.
- Pre-processing, con particolare attenzione alla distribuzione temporale dei dati, alla rilevazione di eventuali outlier e alla definizione delle strategie per il loro trattamento.
- Ricerca di correlazioni tra le variabili indipendenti e la feature target, per identificare i fattori più significativi.
- Divisione del dataset e addestramento dei modelli, con l'applicazione di diverse tipologie di algoritmi per la fase di training.
- Valutazione delle performance dei modelli, analizzando i risultati ottenuti sui dati di test.

2 Analisi del Dataset

Dopo aver importato il dataset utilizzando la libreria *pandas* di Python, procediamo con una prima fase di analisi dei dati presenti.

Index	Column Name	Non-null Count	Data Type
0	Unnamed: 0	62630	int64
1	UTC	62630	int64
2	Temperature[C]	62630	float64
3	Humidity[%]	62630	float64
4	TVOC[ppb]	62630	int64
5	eCO2[ppm]	62630	int64
6	Raw H2	62630	int64
7	Raw Ethanol	62630	int64
8	Pressure[hPa]	62630	float64
9	PM1.0	62630	float64
10	PM2.5	62630	float64
11	NC0.5	62630	float64
12	NC1.0	62630	float64
13	NC2.5	62630	float64
14	CNT	62630	int64
15	Fire Alarm	62630	int64

Tabella 1: Informazioni delle colonne del dataset

Notiamo subito che il dataset presenta soltanto features numeriche rappresentanti i vari valori raccolti dai sensori. L'ultima colonna del dataset, denominata *Fire Alarm*, rappresenta la **target feature**, ovvero una variabile binaria che indica se, in base ai dati presenti nel record attuale, l'allarme antincendio si attivi o meno.
Valutiamo innanzitutto la distribuzione delle due classi all'interno del dataset per verificare che non siano sbilanciate.

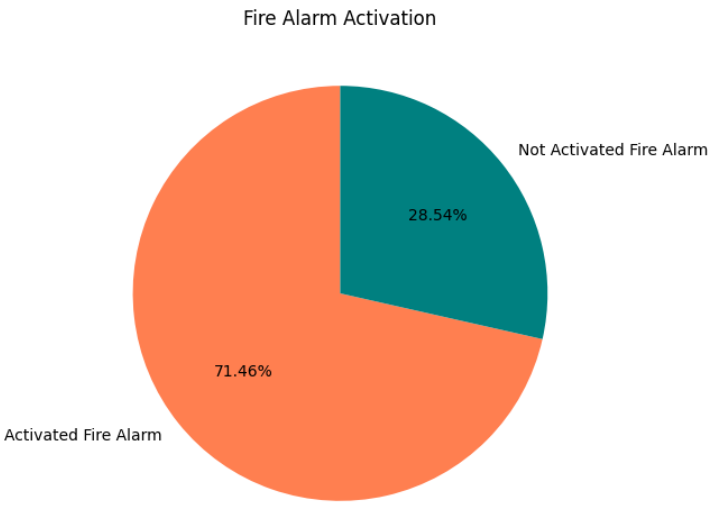

Figura 1: Distribuzione delle classi nel dataset

Osserviamo che nel **71.46%** dei casi l'allarme sia in stato **'Active'** mentre nel rimanente **28.54%** sia nello stato complementare **'Not Active'**.

3 Pre Processing e Data Cleaning

Durante questa fase, che implica una analisi approfondita del dataset, il primo passo consiste nel verificare la presenza di eventuali dati duplicati, che nel nostro dataset si attesta essere pari a zero.

Passiamo dunque alla eliminazione delle colonne considerate irrilevanti per la successiva modellazione, ovvero *Unnamed: 0* e *CNT*, rispettivamente l'id e il counter dei dati inviati da ciascun sensore.

Procediamo quindi con l'analisi della distribuzione dei dati nel tempo. Notiamo dal grafico in Figura 2 che la distribuzione temporale potrà essere un fattore influente sui risultati. Le rilevazioni sono infatti concentrate maggiormente in un solo giorno, dando così una minore diversità dei dati raccolti.

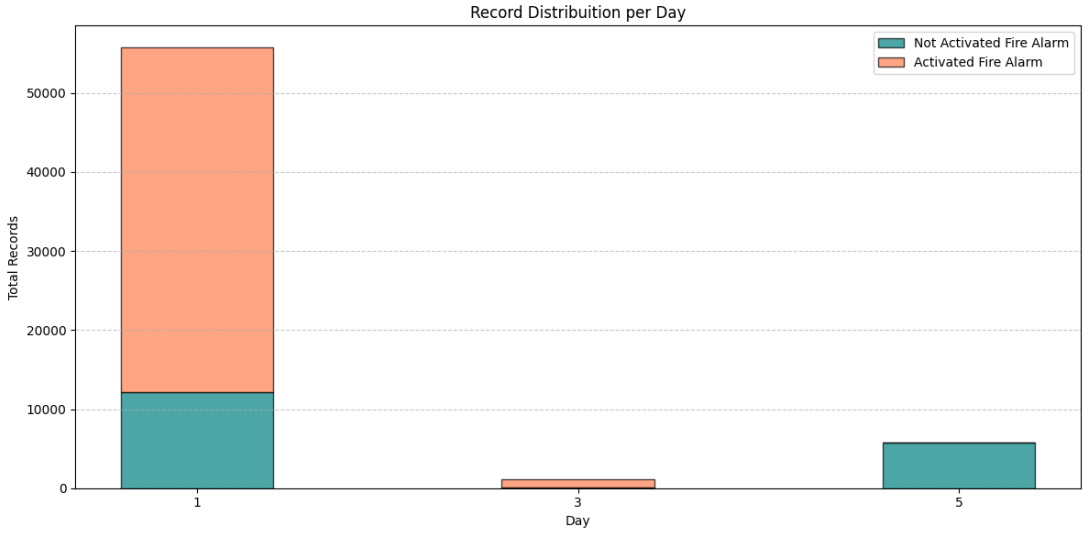

Figura 2: Distribuzione delle classi per giorno.

Questo può portare alla presenza di dati molto simili tra di loro e che possono poi influenzare molto la modellizzazione futura.

Dopo un'analisi più approfondita del dataset, osserviamo che l'attivazione dell'allarme non si concentra in un momento specifico della giornata. Al contrario, sono frequenti i periodi in cui l'allarme si attiva e si spegne, creando così una variazione nei dati raccolti durante lo stesso giorno. Questo fenomeno, quindi, tende a ridurre l'impatto dell'influenza dei dati sulle prossime modellizzazioni.

3.1 Outliers

Nel processo di analisi di un datset qualsiasi, è possibile, inoltre, incorrere nella rilevazione di valori, detti **outliers** che si discostano in maniera significativa dagli altri punti dati e che possono influenzare in maniera negativa i risultati dei modelli in termini di performance.

Per prima cosa, esaminiamo la presenza di outliers nel nostro dataset, analizzando graficamente la distribuzione dei valori di ciascuna feature. A tal fine, utilizziamo istogrammi per ottenere una panoramica preliminare delle feature che potrebbero contenere outliers.

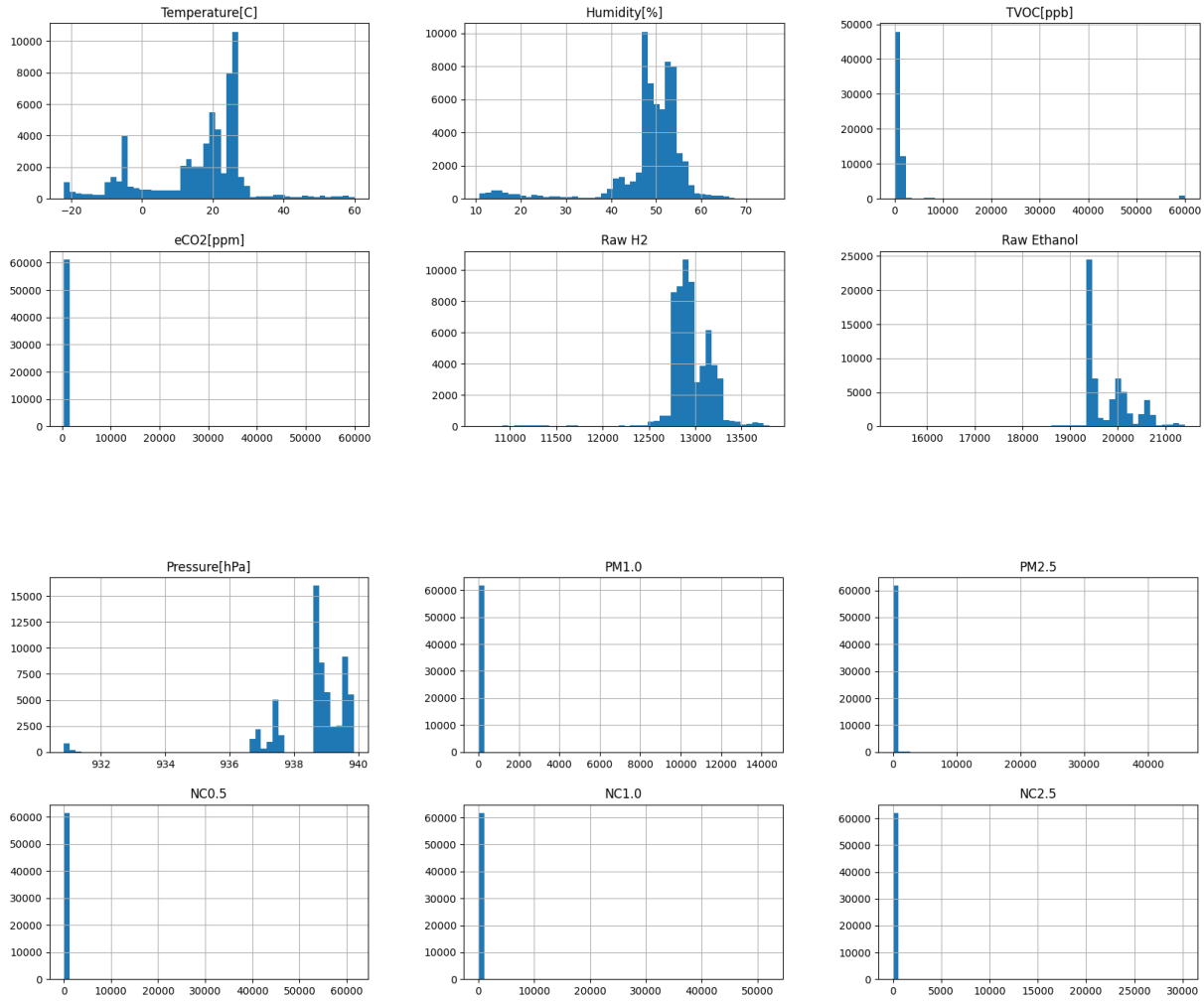

Figura 3: Distribuzione dei valori nel dataset.

Gli outliers sono valori che possono essere generati da varie cause, che spaziano dai semplici errori di misurazione o di inserimento dati a variazioni naturali che noi, non avendo la strumentazione adatta a disposizione non possiamo comprendere.
Questo ci porta dunque ad utilizzare dei metodi statistici per la valutazione degli stessi ed è quello che abbiamo fatto servendoci dell'indice chiamato **Z-Score**.

3.1.1 Z-Score

Lo Z-Score è un indice statistico che viene utilizzato per misurare la distanza di un valore dalla media dei valori in un dataset, in termini di deviazione standard.

Dato quindi un valore x, la media dei dati espressa come μ e la deviazione standard degli stessi dati espressa come σ, si esprime il valore dello Z-Score come:

$$z = \frac{x - \mu}{\sigma}$$

Utilizziamo quindi questo indicatore per rilevare la possibile presenza di valori che si discostano di molto dalla media di quelli presenti nel dataset. Generalmente, dato un intero positivo n (nel nostro caso n=3), si stabilisce che un valore viene considerato outlier se si ha che $|z| > 3$.
Valutando tutte le colonne del dataset, ci si accorge che i possbili outliers non si limitano ad identificare i punti di attivazione dell'allarme antincendio e che esistono delle anomalie in quasi tutte le colonne.

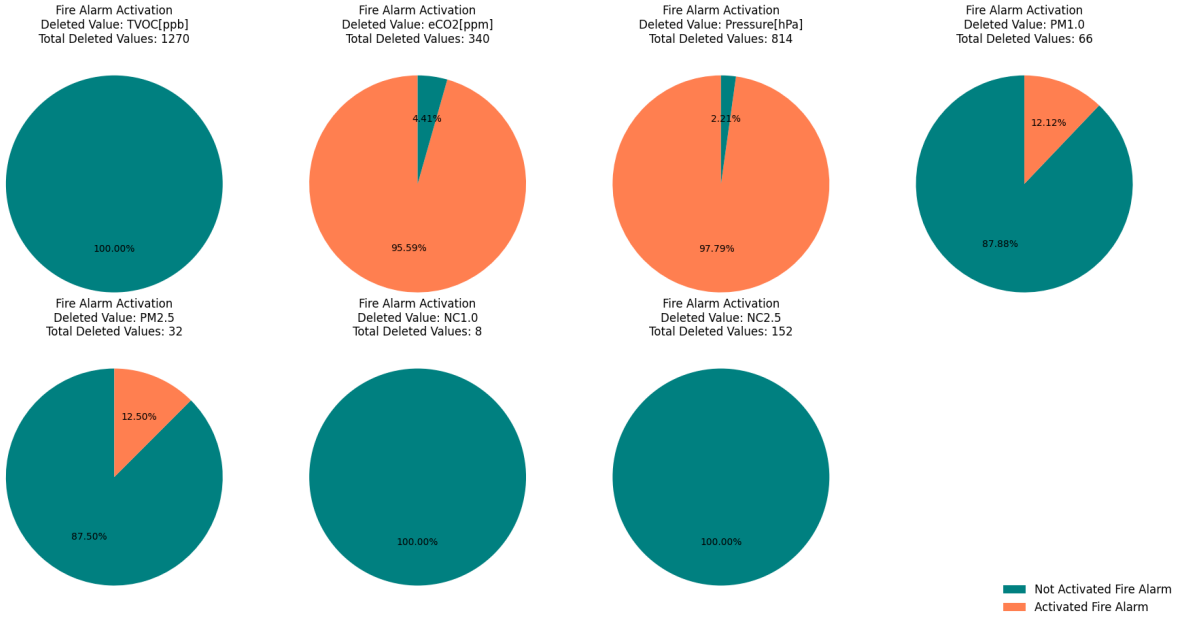

Figura 4: Valori dei possibili outliers con la target feature.

Il passo successivo dunque consiste nel comprendere tra queste colonne, se esistono delle relazioni significative tra di esse. Guardando i risultati prodotti nello studio delle colonne con valori anomali, ci si accorge di come questi rappresentino veri e propri indicatori di una possibile attivazione dell'allarme.

Si nota ad esempio per quanto riguarda la *Pressure(hPa)* di come i valori bassi, che potevano rappresentare degli outliers, in realtà presentino una forte correlazione con il valore di attivazione dell'allarme antincendio.

4 Correllazione tra i dati

Attraverso una rappresentazione grafica ed un analisi feature per feature, abbiamo visualizzato quella che era la correlazione dei vari valori con la generale attivazione dell'allarme, spezzando poi la visualizzazione nelle due classi che rappresentano la target feature.

Temperature

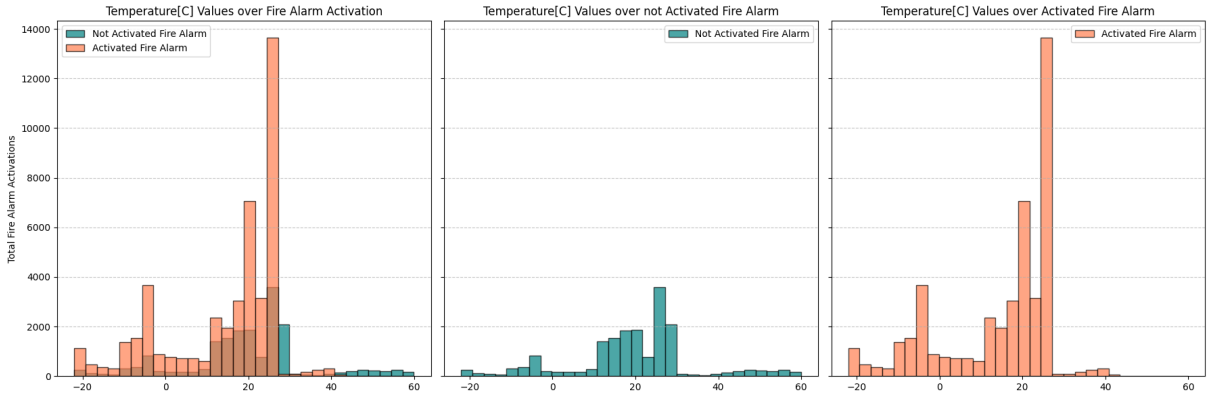

Figura 5: Correlazione della Temperature con target feature.

Humidity

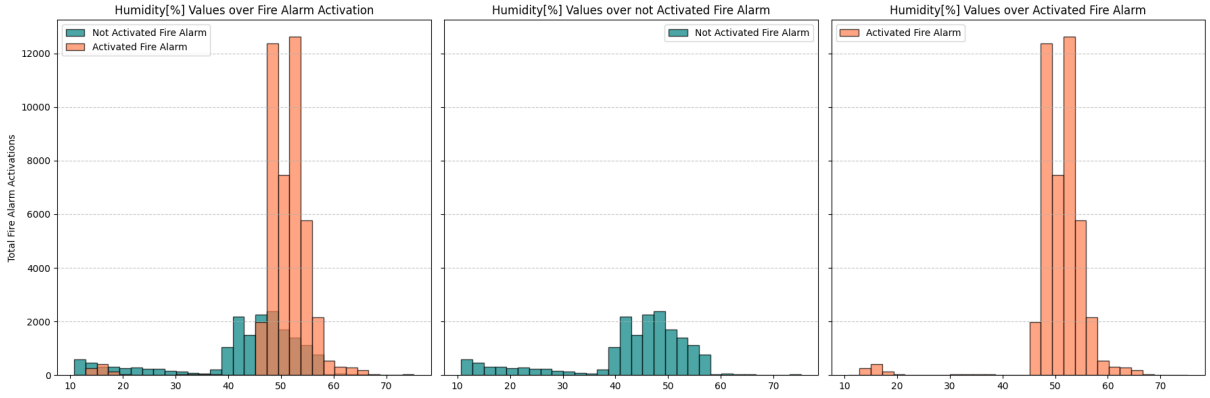

Figura 6: Correlazione della Humidity con target feature.

TVOC

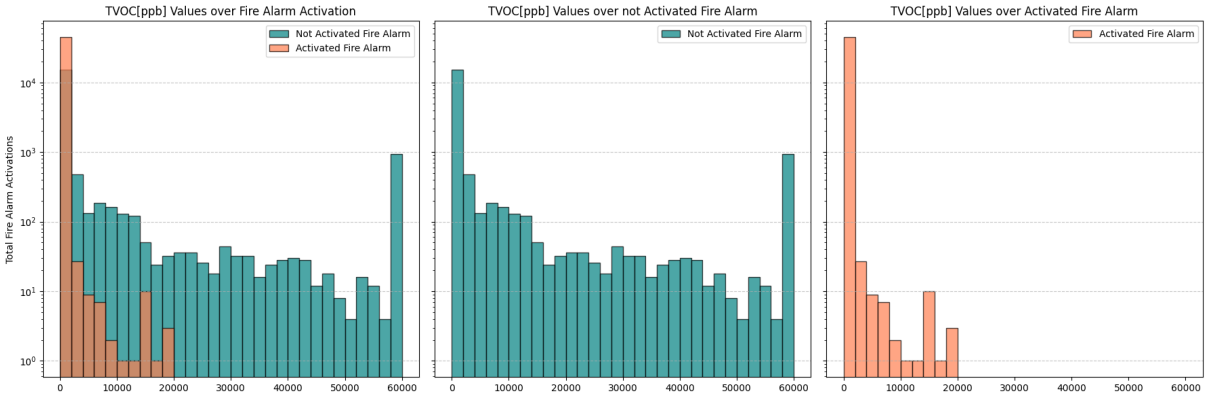

Figura 7: Correlazione di TVOC con target feature.

In alcuni dei grafici adottati, sull'asse delle ordinate si è adottata una scala logaritmica in quanto, sebbene una distribuzione abbastanza lineare dei valori sull'asse delle ascisse (in questo caso si nota che $0 \leq TVOC \leq 60000$), si riesce a notare come in presenza di x≈0 ci siano dei valori di y maggiori. Se avessimo adottato una scala lineare, tali valori sarebbero rimasti maggiormente in vista, a discapito degli altri che sarebbero risultati impercettibili.

Tramite questa scala si riesce ad identificare meglio i valori lungo tutto l'asse delle ascisse, anche se si nota come la maggior parte dei record del dataset abbiano un valore $TVOC \leq 2000$, a discapito di quanto si possa pensare da una prima occhiata al grafico in figura 7.
Questo discorso è ovviamente estendibile anche per quanto riguarda gli altri grafici che presentano una scala logaritmica.

Per brevità mostriamo solo alcuni dei grafici delle feature in relazione con l'attivazione dell'allarme, ovvero quelli che riteniamo più significativi.

eCO2

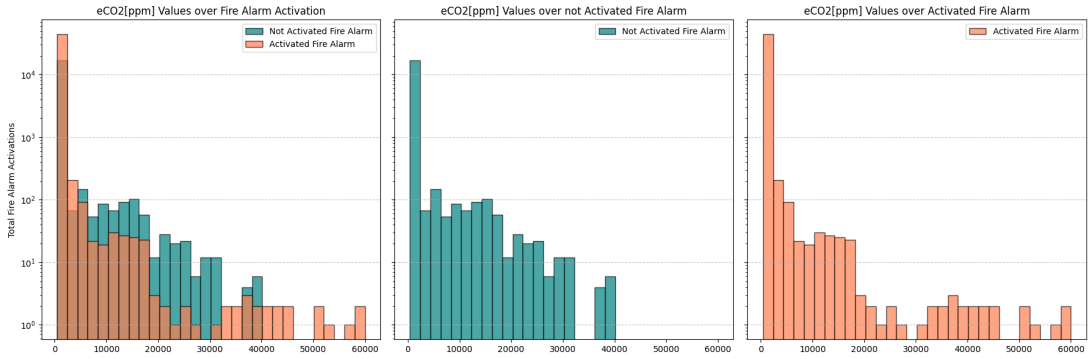

Figura 8: Correlazione della eCO2 con target feature.

Raw Ethanol

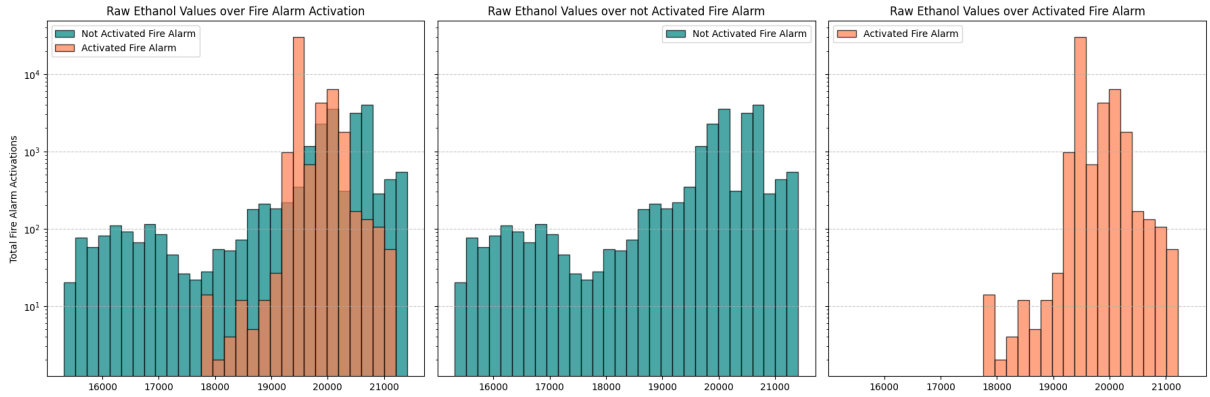

Figura 9: Correlazione di Raw Ethanol con target feature.

4.1 Valore della Pressione

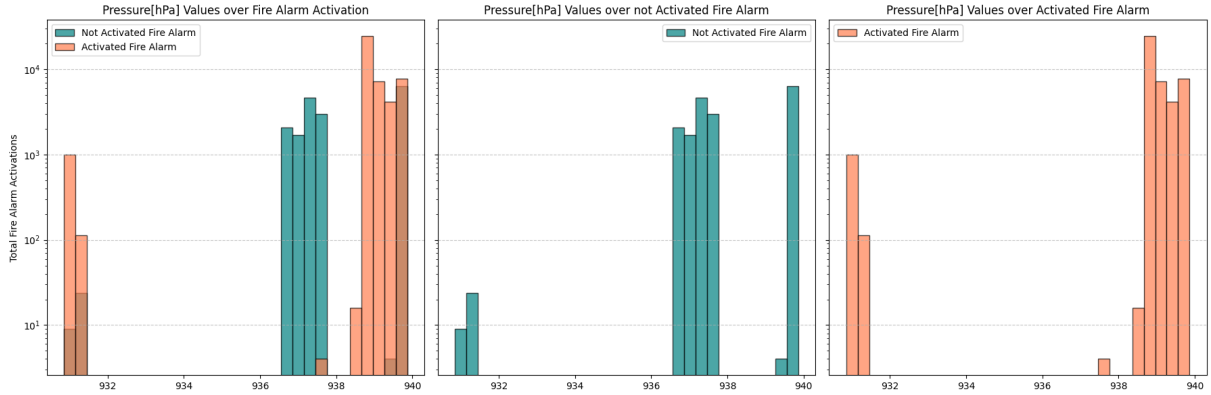

Figura 10: Correlazione della Pressure[hPa] con target feature.

Come si può notare dai grafici in figura 10, si vede una netta separazione tra alcuni dei valori che consentono un'attivazione dell'allarme antincendio.
Possiamo infatti vedere come, dato un valore di pressione p, si ha:

$$p \leq 932 \quad \Rightarrow \quad \text{Fire Alarm} = 1$$

$$936 \leq p \leq 938 \quad \Rightarrow \quad \text{Fire Alarm} = 0$$

$$p > 938 \quad \Rightarrow \quad \text{Fire Alarm} = 1$$

Tramite quest'analisi si può effettuare una deduzione sull'importanza del valore della pressione, identificandolo come un possibile buon indicatore per la target feature.

4.2 Correlation Matrix

Dopo una vista preliminare, attraverso i grafici, della relazione di ciascuna feature con il target, si vuole ora cercare attraverso uno strumento quale la *Correlation Matrix* un riscontro numerico di precisa lettura, in maniera tale da comprendere se e quali di queste features potranno essere particolarmente significative.

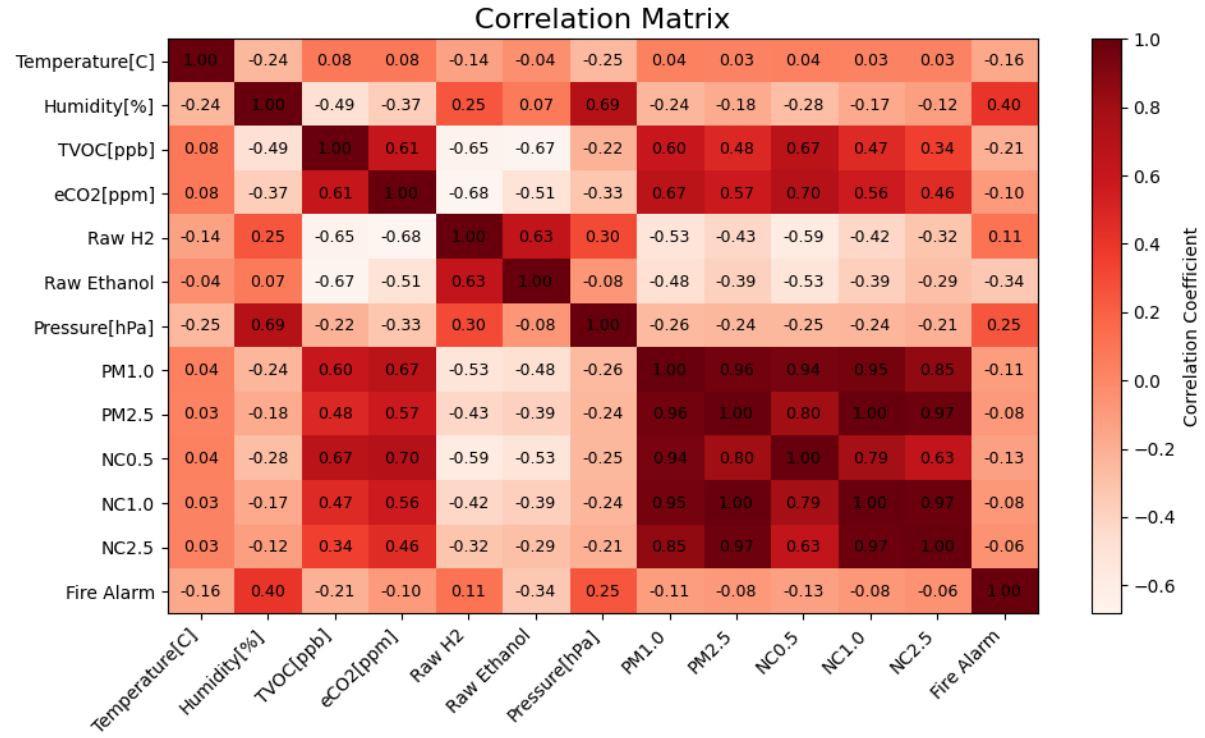

Figura 11: Correlation Matrix delle features.

Se si considera alta una correlazione con un valore superiore allo 0.7, guardando la figura 11, si nota come, tutti i dati del tipo **NC*** e **PM*** risultino fortemente correlati tra di loro, attraverso una tonalità forte del colore utilizzato.

Questo avviene in quanto sono dati riferiti alla concentrazione di particelle simili ma che hanno spessori diversi.

Osservando i grafici nel notebook, si nota che le particelle **PM2.5** sono generalmente presenti in un numero simile, sebbene leggermente superiore, rispetto alle particelle **PM1.0**, che appartengono alla stessa categoria ma hanno un diametro inferiore.

Generalmente, osservando la matrice di correlazione, non emergono valori particolarmente significativi tra le diverse features e il target. Anche nel nostro caso non notiamo particolari correlazioni.

5 Data Split

A questo punto si può procedere con la fase successiva di addestramento dei vari modelli utilizzati.
Per prima cosa si è diviso il dataset in tre diverse porzioni:

- **Training Set** (60%)
- **Validation Set** (20%)
- **Test Set** (20%)

Dopo questa divisione, verifichiamo che la distribuzione della classe target di attivazione dell'allarme antincendio nelle tre diverse porzioni di dataset appena ottenute si mantenga simile.

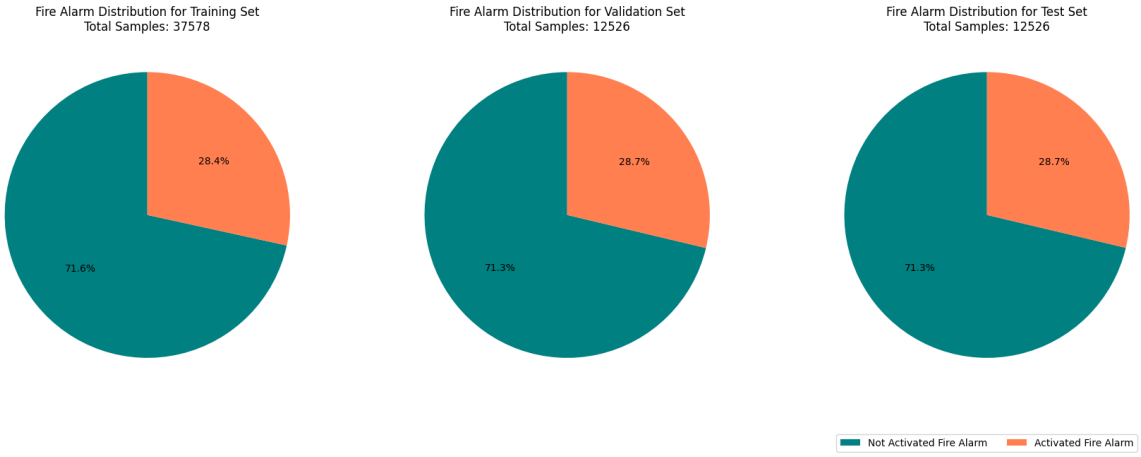

Figura 12: Distribuzione della classe target nei vari set.

6 Model Creation

A questo punto, è il momento di applicare diversi modelli ai dati di training per la fase di apprendimento e successivamente valutarne le prestazioni in termini di tempo e accuratezza. Tale valutazione verrà effettuata utilizzando la porzione di dati destinata al *Test Set*.

Per tenere traccia dei risultati ottenuti durante i vari passaggi con i diversi modelli, è stata sviluppata una classe dedicata. Questa classe consente di raccogliere e conservare tutte le informazioni relative ai modelli utilizzati, come il nome e le performance precedentemente menzionate.

```
class Model():
    model: sk.base.BaseEstimator = None
    train_time: float = None
    accuracy: float = None
    prediction_time: float = None
    # additional information that change from model to model
    add_info: dict = None

    def __init__(self, model: sk.base.BaseEstimator):
        self.model = model

    def __name__(self):
        return self.model.__class__.__name__

    def __str__(self):
        print(f"Model: {self.model.__class__.__name__}")
        print(f"Accuracy: {self.accuracy*100:.2f}%")
        print(f"Training Time: {self.train_time:.6f} seconds")
        print(f"Prediction Time: {self.prediction_time:.6f} seconds")
        if self.add_info:
            for key, value in self.add_info.items():
                print(f"{key}: {value}")

models = {}
```

Nelle successive sezioni saranno indicati quali sono i modelli che sono stati utilizzati e per ciascuno di essi verranno riportate le caratteristiche, come è stato applicato ed i risultati ottenuti.

6.1 Dummy Classifier

Il primo classificatore ad essere testato è un **Dummy Classifier**. Esso non tiene conto delle features in input, ma utilizza una strategia di default: il metodo di predizione ritornerà sempre la classe che presenta la maggiore frequenza tra quelli che sono i valori osservati.

Questo porta ad avere un output con un valore fissato ad 1, ovvero in corrispondenza del valore di maggiore frequenza quando l'allarme antincendio risulta attivato.
Di conseguenza l'Accuracy risulterà del tutto uguale alla percentuale del valore 1 nella target feature del test set, che risulta $\approx 71\%$ I risultati ottenuti sono indicati di seguito e vengono riportati nella classe Model creata.

```
Model: DummyClassifier
Accuracy: 71.31%
Training Time: 0.004371 seconds
Prediction Time: 0.001191 seconds
Unique Values: [1]
Frequency: [12526]
```

6.2 Decision Tree Classifier

Come modello successivo è stato applicato un **Decision Tree Classifier** semplice, ovvero non è stato impostato alcun valore specifico per quanto riguarda gli iperparametri: sono stati utilizzati quelli di default, vale a dire il *GINI* per la valutazione delle impurità e *Best Split* come criterio di splitting in base al valore dell'impurità che è stato rilevato. E' stato inoltre impostato in fase di creazione del modello un *random seed* per garantire la riproducibilità.

Di seguito vengono riportati i risultati ottenuti dal modello ed una rappresentazione grafica completa dell'albero.

```
Model: DecisionTreeClassifier
Accuracy: 99.97%
Training Time: 0.095864 seconds
Prediction Time: 0.001550 seconds
Max Depth: 8
Number of Nodes: 28
```

Come era già stato presentato nella sezione 4.1, il valore della Pressione potrebbe rappresentare una feature particolarmente utile per effettuare la classificazione.

Osservando l'albero di decisione[1] infatti, si nota come il nodo rappresentante la radice si divida in corrispondenza del valore di 938.143 hPa. Questo risultato conferma quanto avevamo già visto in precedenza attraverso la rappresentazione grafica attraverso gli istogrammi. Analizzando la features importance dell'albero decisionale appena modellato, presente in figura 13, si osserva che la Pressione è quella che contribuisce maggiormente, registrando la percentuale più alta tra tutte.

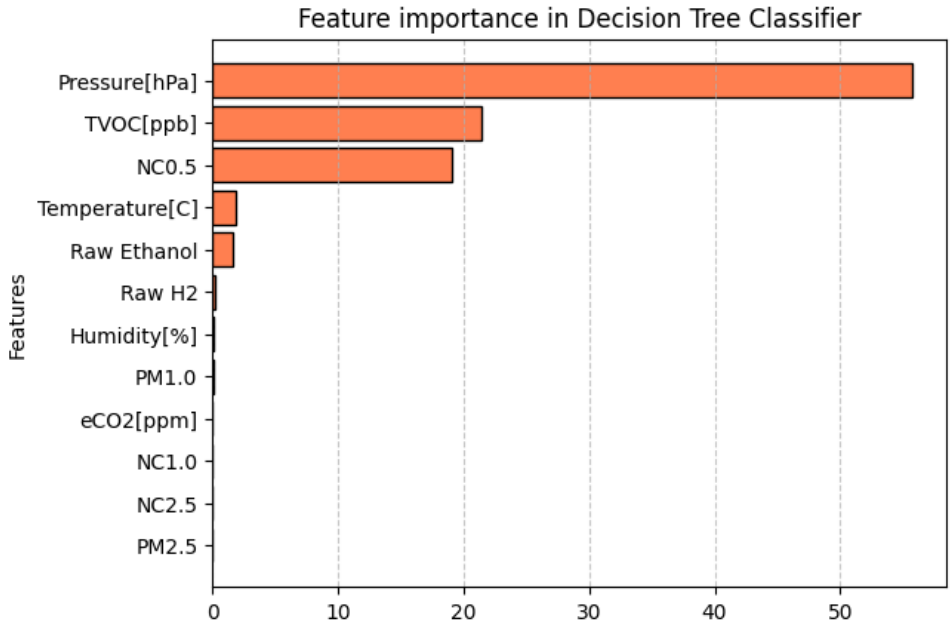

Figura 13: Feature importance del Decision Tree

Come evidenziato dai risultati riportati sopra, l'albero modellato mostra un elevato valore di Accuracy.

Questo risultato, come anticipato durante lo studio del dataset, è probabilmente influenzato dalla distribuzione dei dati. I dati, infatti, sono prevalentemente concentrati in un singolo giorno e non presentano outlier.

La disponibilità di un pattern ben definito e omogeneo, unita alla chiara separabilità tra le classi, favorisce una rappresentazione efficace da parte del Decision Tree.

[1]Presente nel notebook

6.2.1 Modifica della profondità

Si vuole adesso provare a variare il valore degli iperparametri, modificando ad esempio il numero di nodi dell'albero, aumentando dunque quella che è la sua profondità massima.

L'obiettivo di questa variazione è verificare se la modifica introdotta ha causato il fenomeno di **Overfitting**, analizzando una possibile divergenza tra l'Accuracy del Training Set e quella del Validation Set.

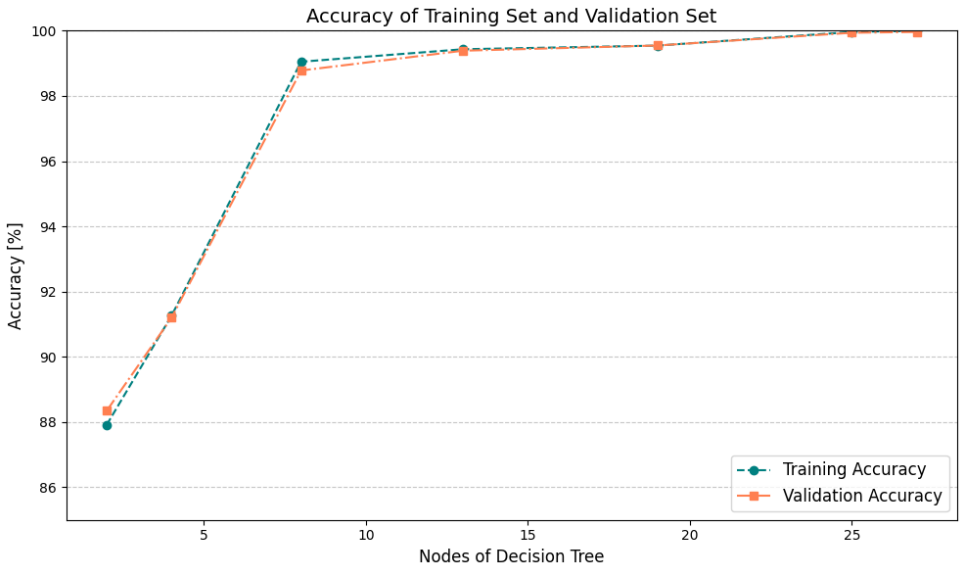

Figura 14: Accuracy nel Decision Tree

L'omogeneità nella distribuzione dei dati e l'assenza di outliers significativi, come già riportato, porta il modello ad un valore di accuracy di quasi il 100%.
Essendovi la medesima distribuzione all'interno del Training Set, Validation Set e Test Set, non avviene il fenomeno dell'overfitting, che avverrebbe normalmente con una diversa distribuzione dei dati; al contrario Training Set e Validation Set vanno di pari passo verso un valore di Accuracy sempre maggiore e tendente al 100%, vista anche l'abilità dell'albero nella accurata suddivisione dei valori delle feature.

6.3 Random Forest Classifier

Il Random Forest Classifier introduce un elemento di randomizzazione nella selezione delle features, con l'obiettivo di migliorare la robustezza del modello e ridurre il rischio di overfitting rispetto a un singolo albero decisionale. In questa sezione si intende analizzare se questa tecnica produce risultati significativamente differenti rispetto a quelli ottenuti in precedenza.
Inseriamo di seguito i risultati ottenuti:

```
Model: RandomForsetClassifier
Accuracy: 100.00%
Training Time: 3.643291 seconds
Prediction Time: 0.037854 seconds
Number of Estimators: 100
```

Notiamo, da un ulteriore analisi, che nonostante diminuiamo il numero di estimators, che di default sono pari a 100, il valore dell'accuracy del modello non diminuisce ma resta fisso al 100%.

Visualizziamo anche in questo caso i valori relativi alla feature importances in maniera da poterli confrontate con quelli del descision tree appena valutato.

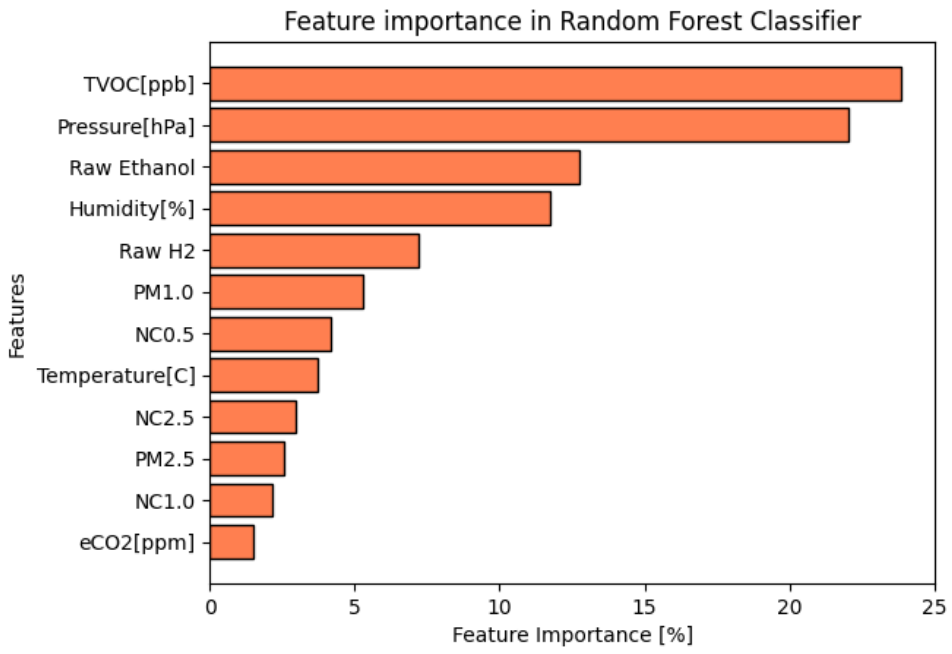

Figura 15: Feature importance Random Forest

Notiamo che la percentuale di influenza delle singole variabili è molto più proporzionata rispetto al modello precedente.

Una Random Forest è infatti un insieme di alberi decisionali, che vengono costruiti su sottoinsiemi casuali dei dati e delle features.

Ogni albero considera soltanto un sottoinsieme delle feature ad ogni split, riducendo il rischio che una sola feature dominante possa influenzare il modello.

Questo porta ad una distribuzione più equilibrata dell'importanza tra le varie feature. La Feature Importance nelle Random Forest viene calcolata nel seguente modo:

$$fi_j^{(k)} = \frac{\sum_{i:\text{nodo } i \text{ splitta su attributo } j} ni_i^{(k)}}{\sum_{i \in nodes} ni_i^{(k)}}$$

$$norm\ fi_j^{(k)} = \frac{fi_j^{(k)}}{\sum_{i \in features} fi_i^{(k)}}$$

$$RF\ fi_j = \frac{\sum_{k\ \in Decision\ Trees} norm\ fi_j^{(k)}}{Number\ of\ Trees}$$

Questa media permette di ridurre il valore della varianza, ma può portare ad una diluizione dell'importanza delle feature che sono fortemente influenti nei singoli alberi.

I *Decision Tree* al contrario, tendono a favorire quelle features che hanno una forte correlazione con il target, come visibile in figura 13.

Dato che non viene effettuato un campionamento casuale, l'albero risulta deterministico e l'importanza delle feature dipende interamente dalla struttura dell'albero ottimizzato sui dati di training.

6.4 Bagging Classifier

Il **Bagging Classifier** condivide molte somiglianze con il Random Forest, ma si distingue per l'assenza del sampling delle features, concentrandosi invece solo sul bagging dei dati. Questo approccio mira a ridurre la varianza del modello, mantenendo una maggiore fedeltà alle caratteristiche originali del dataset. Applicando il modello ai dati a disposizione, i risultati ottenuti sono i seguenti:

```
Model: BaggingClassifier
Accuracy: 99.98%
Training Time: 0.963204 seconds
Prediction Time: 0.010879 seconds
Number of Estimators: 10
```

Arrivati a questo punto del lavoro, è evidente che, per il dataset in uso, variare il numero di estimators non provoca modifiche nella classificazione finale del modello né nell'accuratezza ottenuta.

Possiamo quindi concentrarci di nuovo sulla feature importance.

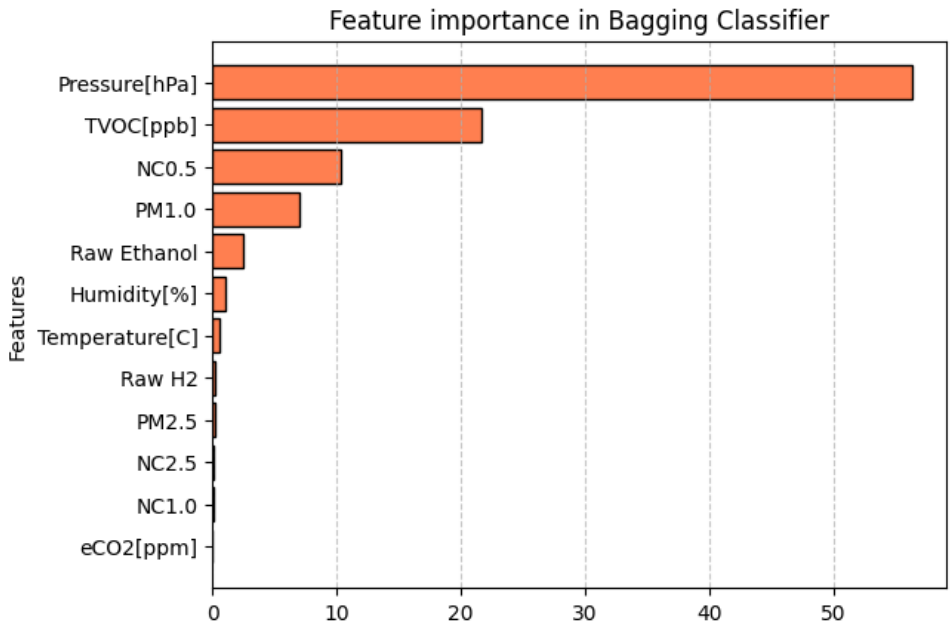

Figura 16: Feature importance Bagging Classifier

Quello che si evince dalla figura 16 è che i dati sono molto simili a quelli riportati in figura 13 e riguardanti la feature importance del Decision Tree.

Le ragioni di questa similitudine sono principalmente legate al fatto che il Bagging Classifier non esegue una selezione delle features e si compone di una serie di Decision Tree, il che lo rende molto simile al risultato ottenuto nella valutazione precedente.

6.5 Boosting Classifier

In questa fase, viene testato il **Boosting**, un modello iterativo che si distingue dai precedenti per la sua capacità di migliorare progressivamente le performance. Questo avviene attraverso la combinazione di più modelli deboli per creare un predittore complessivo più forte.L'idea di base è correggere gli errori commessi dai modelli precedenti, assegnando maggiore peso ai campioni classificati erroneamente.

Il modello scelto per questa analisi è **AdaBoost**, le cui performance sono mostrate di seguito.

```
Model: AdaBoostClassifier
Accuracy: 99.72%
Training Time: 1.490747 seconds
Prediction Time: 0.025724 seconds
Number of Estimators: 50
```

Le features importance sono simili a quelle ottenute in precedenza, in quanto AdaBoost, come il Bagging, non esegue il campionamento delle feature.

6.6 Support Vector Machines

Il prossimo modello che viene utilizzato per testare le performance sui dati è **Support Vector Machine**.

Quello che viene verificato è se la separabilità delle feature trovata attraverso i metodi precedenti, in particolare gli alberi decisionali, è attuabile sfruttando gli *Iperpiani Separatori* utilizzati da SVM.

Questi ultimi possono risultare molto efficienti nel nostro caso in cui dobbiamo eseguire una Binary Classification.

Le SVM (Support Vector Machines) sono utilizzate per massimizzare la separazione tra le due classi, cercando di trovare l'iperpiano che ottimizza il margine tra i dati, al fine di ottenere buoni valori di accuratezza sul dataset.

L'approccio seguito consiste nell'utilizzare le *Kernel Function*, che permettono di mappare i dati in uno spazio di dimensioni superiori. Attraverso diverse tipologie di kernel, come il *linear*, *polynomial* o *RBF*, che vengono passate come parametro delle funzioni utilizzate, è possibile migliorare la separabilità delle classi e ottenere migliori prestazioni nel modello SVM.

Dopo una attenta valutazione delle performance di ciascuno di essi, valutate con le 12 features, viene effettuata una riduzione delle dimensioni in maniera tale da riuscire a visualizzare graficamente la bontà dei risultati in due dimensioni su un grafico.

6.6.1 Linear Kernel

Il primo tipo di kernel che viene valutato è il kernel lineare. In questo caso, considerando due dati appartenenti a classi differenti, se essi sono linearmente separabili, viene calcolato il prodotto scalare tra i due vettori, secondo la seguente espressione:

$$K(X_i, X_j) = X_i \cdot X_j$$

Successivamente vengono presentati i risultati ottenuti, con una visione di insieme sulle dimensioni standard e dopo aver effettuato il processo di riduzione delle dimensioni.

```
-- Standard Dimensions --
Model: SVC
Accuracy: 90.68%
Training Time: 12.452087 seconds
Prediction Time: 1.263674 seconds
Scale: True
Train Ratio: 1.0
None
```

```
-- Dimensionality Reduction --
Model: SVC
Accuracy: 73.04%
Training Time: 4.451574 seconds
Prediction Time: 1.712622 seconds
Scale: True
Train Ratio: 0.4
None
```

Volendo vedere graficamente la separazione delle due classi, con un indicazione su quali siano i valori che vengono considerati come **Support Vector**, si può plottare un grafico come quello sottostante.

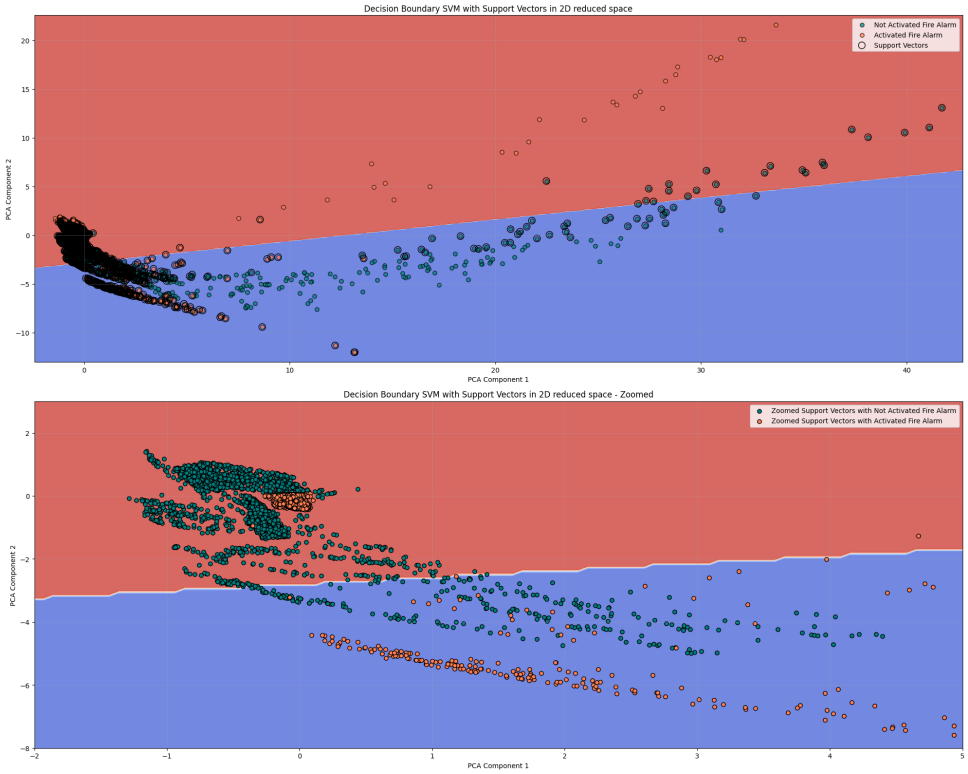

Figura 17: Iperpiano separatore con Kernel Lineare

6.6.2 Polynomial Kernel

La seconda tipologia di kernel che è stato utilizzato è il **Polynomial Kernel**, il quale a differenza del precedente, permette di catturare facilmente delle relazioni che non sono lineari tra i dati in maniera efficace, attraverso un'espressione come la seguente:

$$K(X_i, X_j) = (p + X_i \cdot X_j)^q$$

in cui:

- p è un termine additivo al prodotto scalare tra i due elementi. Di default si ha p = 'auto' da sklearn, ovvero $p = \frac{1}{|features|}$
- $q > 0$ è il grado del polinomio che viene utilizzato (di default si ha q=3)

Anche in questo caso vengo riportati i risultati ottenuti tramite l'applicazione di questo modello successivamente:

```
-- Standard Dimensions --
Model: SVC
Accuracy: 91.12%
Training Time: 9.544609 seconds
Prediction Time: 1.481914 seconds
Scale: True
Train Ratio: 1.0
None
```

```
-- Dimensionality Reduction --
Model: SVC
Accuracy: 72.71%
Training Time: 20.258171 seconds
Prediction Time: 1.864328 seconds
Scale: True
Train Ratio: 0.4
None
```

Volendo visualizzare quelli che sono i risultati in termini di separazione tra le due classi, mostriamo il plot del grafico risultante:

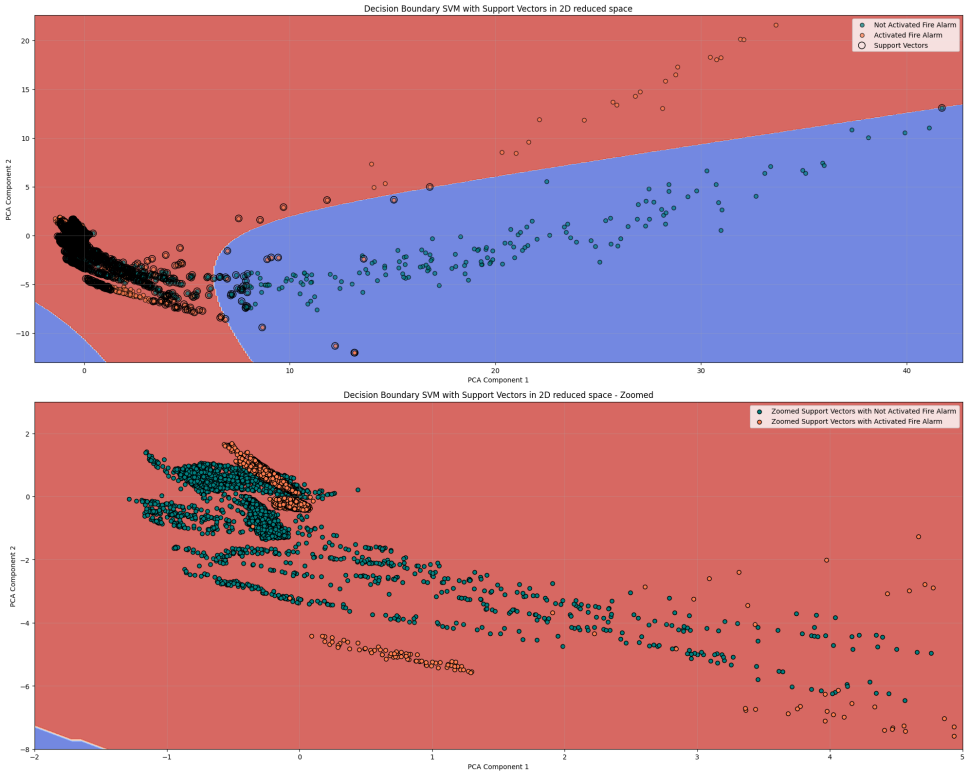

Figura 18: Iperpiano separatore con Kernel Polinomiale.

6.6.3 Kernel Gaussiano

Il **Kernel gaussiano**, anche noto come *Radial Basis Function* (RBF) kernel, è uno dei kernel più utilizzati per catturare relazioni non lineari tra i dati. La sua forza risiede nella capacità di mappare i dati in uno spazio di dimensioni teoricamente infinite, permettendo una separabilità più flessibile e potente, anche in presenza di strutture complesse nei dati.

L'espressione che lo contraddistingue è la seguente:

$$K(X_i, X_j) = exp(-\gamma \ ||X_i - X_j||^2)$$

in cui si ha che è presente:

- $||X_i - X_j||^2$ è la distanza euclidea tra i due punti.
- γ è un parametro di scala che permette di controllare l'influenza di un singolo punto di training.

Anche in questo caso andiamo a valutare le solite performance come nei casi precedenti, ponendo a confronto prima e dopo la riduzione delle dimensioni e mostrando la separazione tra le due classi.

```
-- Standard Dimensions --
Model: SVC
Accuracy: 96.67%
Training Time: 6.556522 seconds
Prediction Time: 1.809109 seconds
Scale: True
Train Ratio: 1.0
None
```

```
-- Dimensionality Reduction --
Model: SVC
Accuracy: 89.83%
Training Time: 2.132744 seconds
Prediction Time: 2.376420 seconds
Scale: True
Train Ratio: 0.4
None
```

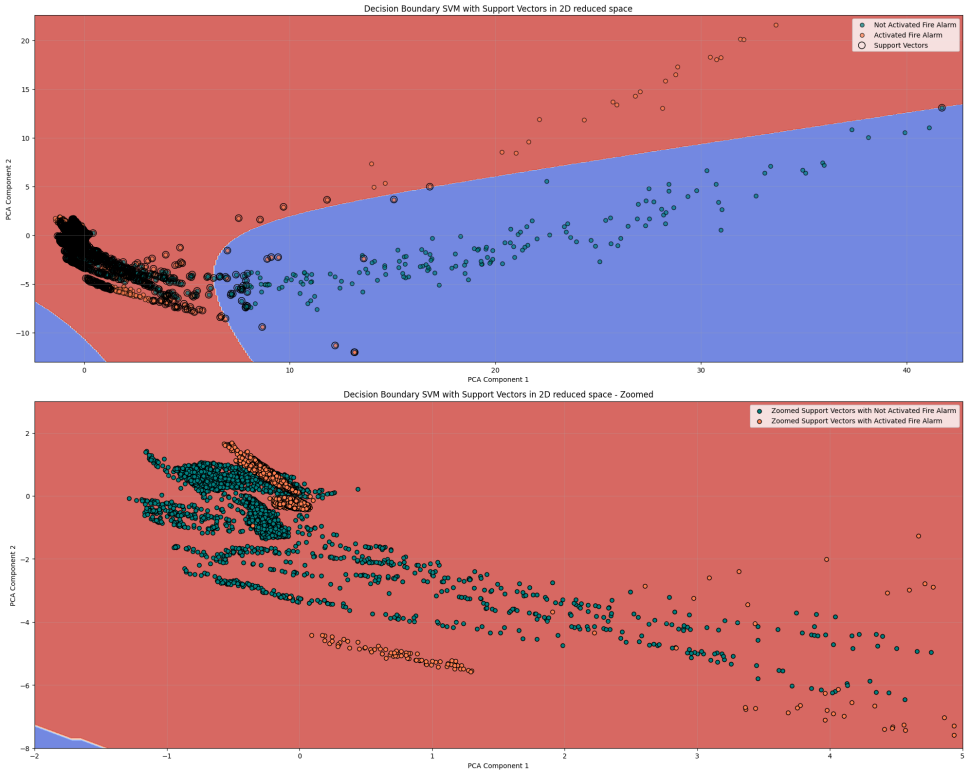

Figura 19: Iperpiano separatore con Kernel Gaussiano

6.6.4 Kernel Sigmoidale

Si tratta di un kernel per catturare relazioni non lineari tra i dati, anche se per il dataset adottato non risulta molto efficace essendoci relazioni tra i dati di carattere lineare.
E' un kernel ispirato dalla funzione di attivazione *sigmoide* utilizzata nelle reti neurali.
L'espressione che lo caratterizza e che viene utilizzate è la seguente:

$$K(X_i, X_j) = tanh(kX_i \cdot X_j - \delta)$$

Successivamente vengono presentati i risultati e mostrata la separazione dei dati con l'iperpiano separatore in due dimensioni.

```
-- Standard Dimensions --
Model: SVC
Accuracy: 68.87%
Training Time: 26.609160 seconds
Prediction Time: 4.437173 seconds
Scale: True
Train Ratio: 1.0
None
```

```
-- Dimensionality Reduction --
Model: SVC
Accuracy: 61.72%
Training Time: 6.383549 seconds
Prediction Time: 2.817419 seconds
Scale: True
Train Ratio: 0.4
None
```

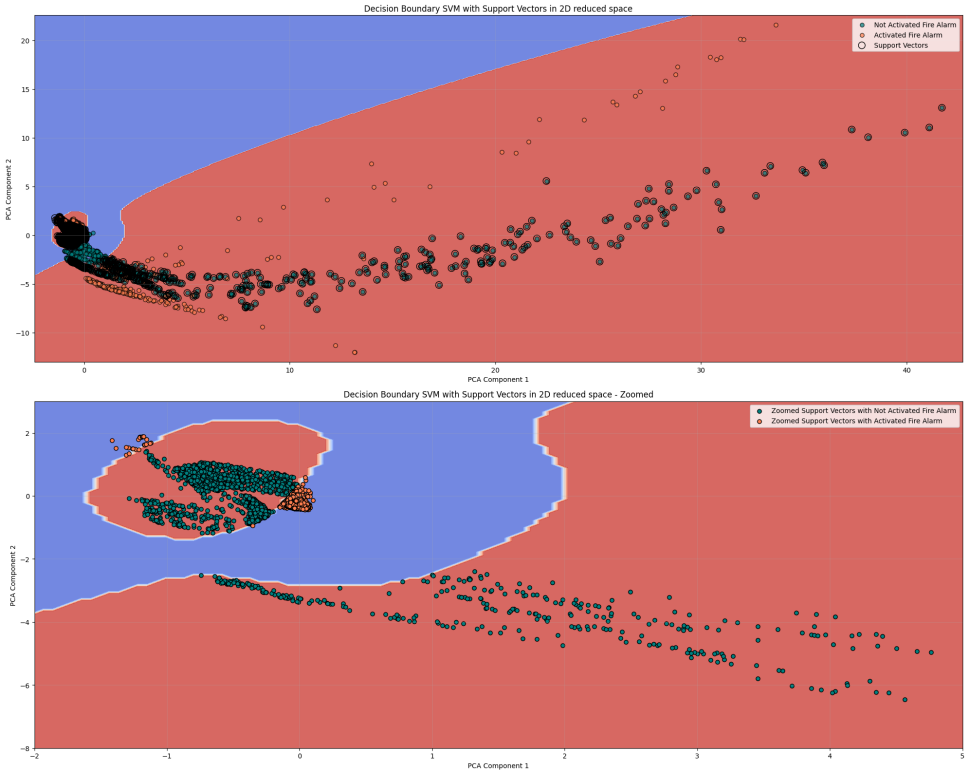

Figura 20: Iperpiano separatore con Kernel Sigmoidale

6.7 Neural Network

Come si è già visto dall'applicazione dei modelli precedenti di Machine Learning, le feature possono risultare ben seperabili attraverso l'utilizzo dei Decision Tree.

Quello che si suppone dunque è che una semplice rete neurale possa sortire lo stesso effetto, permettendoci di ottenere buoni risultati.

L'utilizzo delle librerie python necessarie per operare con le reti neurali avviene attraverso l'inclusione di *tensorflow* con *keras*.

Il primo modello di rete neurale che viene testata è il più semplice possibile ed anche i successivi si baseranno sull'utilizzo di una classe che consente le operazioni base di costruzione ed invocazione. Come prima fase occorre preparare i dati scalandoli ed utilizzando l'output **one-hot encoded** per stimare le probabilità della target feature attraverso la funzione **softmax**. Per iniziare sono stati selezionati l'ottimizzatore **SGD**, la funzione di perdita **Binary Cross-Entropy**, dato che stiamo facendo un processo di classificazione binaria, e come ultima scelta prendiamo la metrica **Binary Accuracy**.

Ogni test delle reti neurali con qualsiasi parametro viene eseguito su un numero di 10 epoche.

Di seguito vengono riportati i risultati della prima semplice rete neurale:

```
Model: SimpleNeuralNetwork
Accuracy: 70.14%
Training Time: 9.594450 seconds
Prediction Time: 0.429832 seconds
```

Questo modello mostra performance mediocri, perciò passiamo subito ad una corretta regolazione di alcuni iperparametri che pensiamo possano portare a risultati migliori.

Le principali strategie che si possono adottare sono le seguenti:

- **Dropout:** durante il training ad ogni iterazione, il dropout standard consiste nell'azzerare una frazione dei nodi in ogni layer prima di calcolare il layer successivo, così da evitare di favorire alcuni collegamenti della rete neurale a scapito di altri, e distribuendo più uniformemente i nodi ed i collegamenti rilevanti.

- **Numero di layer:** possiamo aumentare il numero layer nascosti per modellare funzioni più complesse che possano adattarsi meglio alla nostra funzione target.

- **Layer di output con funzione sigmoid:** la funzione sigmoid può essere utilizzata come layer di output singolo. Attualmente stiamo usando softmax per ottenere l'intera distribuzione (un layer di output con due elementi: uno per *p0=P(Fire Alarm=0)* e uno per *p1=P(Fire Alarm=1)*); possiamo usare un output sigmoid, indicando solo p1 , e quindi calcolare *p0 = 1 - p1*

- **Funzioni di attivazione diverse:** possiamo utilizzare funzioni di attivazione diverse per i layer nascosti, come TANH o RELU, invece di SIGMOID.

- **Ottimizzatore diverso:** possiamo provare un ottimizzatore differente, come ADAM.

La funzione di perdita attuale e la metrica di accuratezza sono già molto adatte al nostro problema, quindi scegliamo di mantenerle.
Proviamo a implementare una classe che utilizza i seguenti iperparametri:

- **Dropout:** 0.2
- **3 layer nascosti** completamente connessi con rispettivamente 128, 64 e 32 neuroni.
- **1 layer di output** con SIGMOID che rappresenta *p1*.
- **Funzione di attivazione** TANH: proviamo ad usare una funzione di attivazione diversa.
- **Adam Optimizer** come ottimizzatore differente.

Prima Implementazione

Effettuiamo quindi la prima implementazione di una generica rete neurale che prevede l'utilizzo della *sigmoide* come funzione di attivazione.

Mostriamo di seguito i risultati ottenuti applicando la rete neurale presentata:

```
Model: GenericNeuralNetwork
Accuracy: 99.48%
Training Time: 12.570954 seconds
Prediction Time: 0.387834 seconds
```

Dal riquadro riportato, si nota che questo modello migliora significativamente le performance del modello precedente.

Seconda implementazione

Per una seconda implementazione, si sceglie di utilizzare una funzione di output *softmax*. Anche in questa nuova configurazione viene utilizzata la funzione di attivazione precedentemente riportata. Vengono inoltra mantenuti i parametri impostati durante il processo.

I risultati ottenuti con questa rete sono i seguenti:

```
Model: GenericNeuralNetwork
Accuracy: 99.22%
Training Time: 12.611184 seconds
Prediction Time: 0.435266 seconds
```

Quello che si nota è il fatto che non ci sono particolari differenze la rete neurale precedente.

Terza Implementazione

Proviamo quindi con una terza implementazione. In questo caso cambiamo la funzione di attivazione, usando la funzione di *relu*, sempre sulle 10 epoche di test precedentemente citate.

Presentiamo dunque i risultati ottenuti:

```
Model: GenericNeuralNetwork
Accuracy: 99.35%
Training Time: 12.144992 seconds
Prediction Time: 0.438851 seconds
```

7 Conclusioni

Una volta applicati tutti i modelli citati nella precedente sezione, i dati relativi all'accuratezza del modello sul test set, il tempo di addestramento del modello ed il tempo necessario per le operazioni di testing vengono settati come attributi della classe usata per rappresentare i vari modelli.
A questo punto, sempre utilizzando dei grafici possiamo trarre delle conclusioni osservando quelli che sono i valori appena citati come barre di istogrammi, in maniera tale da riuscire a fare dei confronti oggettivi sui singoli modelli.

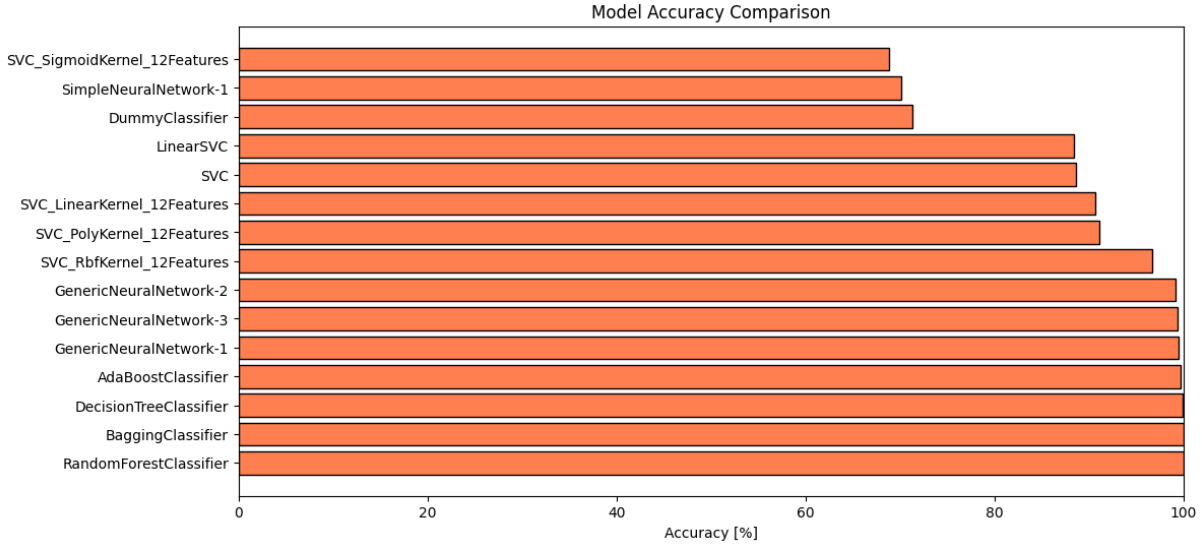

Figura 21: Accuracy Comparison

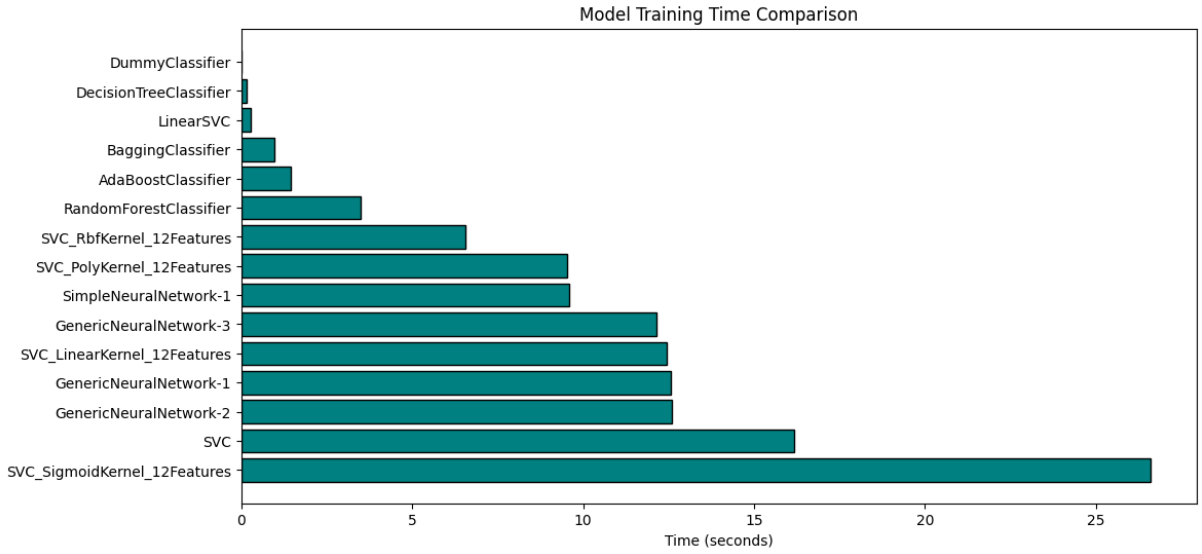

Figura 22: Training Time Comparison

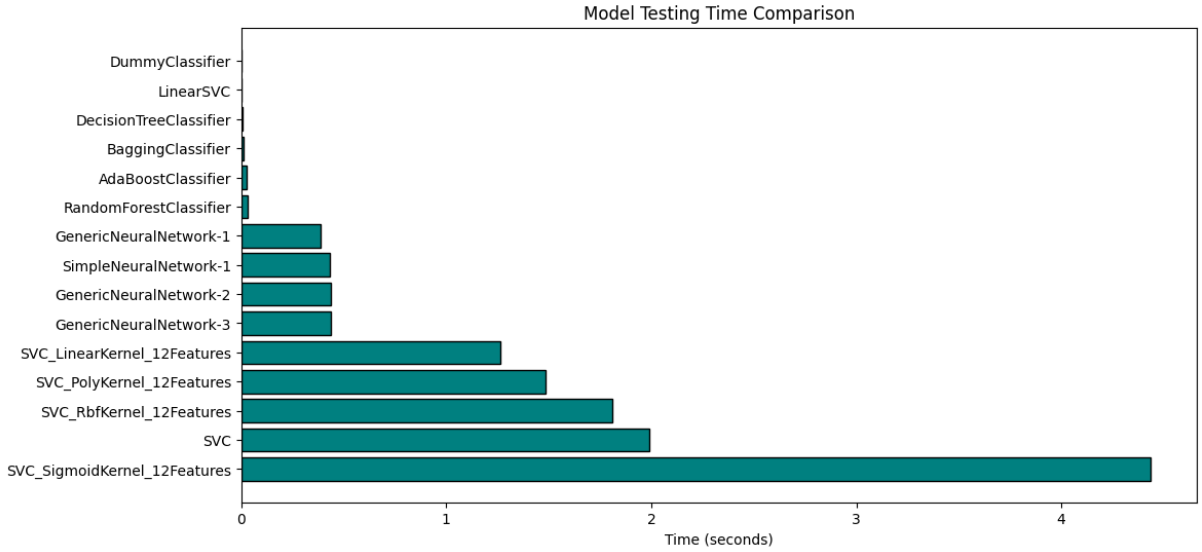

Figura 23: Testing Time Comparison

Nel nostro caso, i metodi basati su alberi decisionali, come Decision Tree e Random Forest, e più in generale gli approcci di tipo ensemble, risultano particolarmente indicati. Questi modelli offrono un ottimo rapporto tra accuratezza e risorse computazionali richieste: raggiungono quasi il 100% di accuratezza e richiedono tempi di addestramento ridotti.

Considerando la distribuzione dei dati del nostro dataset, anche reti neurali leggermente più avanzate possono offrire ottime prestazioni, sebbene potrebbero risultare eccessive per le caratteristiche del dataset. Al contrario, le Support Vector Machines (SVM) si so-

no dimostrate meno efficaci, sia per i tempi di elaborazione elevati sia per un'accuratezza inferiore rispetto agli altri modelli.

Questa analisi sottolinea che, in relazione alla struttura e alla qualità dei dati disponibili, i metodi ensemble rappresentano la scelta ottimale.

Note implementative

Il progetto è stato realizzato utilizzando il linguaggio Python, il tempo totale di esecuzione del notebook è di **326.58 secondi**, considerando come processore di riferimento *11th Gen Intel(R) Core(TM) I7-1165G7 @ 2.80GHz*.