Corso di Intelligenza Artificiale

Laurea Magistrale in Informatica

Università di Bologna

Anno accademico 2022/2023

Gruppo: mAIntenance

Abruzzese Michele [0001097676]

Cannas Elia [0001097520]

Rossi Emanuele [0001099633]

Spadoni Luca [0001099632]

Indice

1 Introduzione

A seguito della sempre più alta propensione delle imprese a perseguire il cosiddetto modello di Industria 4.0, il bisogno di avere un sistema in grado di determinare se un particolare macchinario possa incorrere o meno in un malfunzionamento e di riconoscere quest'ultimo, è considerato essenziale. I vantaggi legati all'utilizzo di un sistema di questo tipo sono principalmente di natura economica: infatti, la sostituzione di un singolo componente di un macchinario difettoso, comporta spese estremamente minori rispetto a quelle relative alla sostituzione totale del macchinario stesso. Perciò sono stati introdotti dei sensori, che una volta installati sulle macchine, permettono di monitorarne lo stato mediante una raccolta di dati specifici e appropriati.

Il nostro lavoro consiste in un'analisi specifica dei dati raccolti dai sensori per lo sviluppo di un applicativo di Machine Learning, che sfrutta le correlazioni scoperte nell'insieme dei dati per fornire una predizione sufficientemente valida e corretta riguardo ad un probabile guasto di un macchinario.
Per prima cosa abbiamo eseguito un lavoro di raffinamento e di preprocessing del dataset utilizzato, ovvero il "AI4I Predictive Maintenance Dataset" ottenuto dal Repository UCI, in modo tale da tenere i dati utili allo scopo e scartare tutto il resto. Successivamente, abbiamo eseguito un'analisi esplorativa dei dati (EDA), ovvero uno studio approfondito del dataset volto a scoprirne le principali caratteristiche e a ricercare eventuali pattern, utilizzando strumenti di analisi statistica. Infine, abbiamo applicato alcuni algoritmi di classificazione per capire quale tra essi riuscisse a svolgere il lavoro con la maggiore precisione. Siamo riusciti a sviluppare un modello in grado di fornire un output accettabile, che restituisce predizioni con un tasso di accuratezza prossimo al 100%.

L'iter da noi seguito risulta essere pressappoco lo stesso di quello di alcune implementazioni esistenti che sono state trovate su Kaggle. Quest'ultimo è una piattaforma che permette agli utenti di trovare e pubblicare datasets, esplorare e costruire modelli in un ambiente online, fornendo potenza di calcolo virtuale aggiuntiva e di proporre possibili soluzioni a problemi preesistenti. I lavori da noi osservati e presi in considerazione per questa implementazione risultano seguire una traccia comune, con le uniche significative differenze nella gestione del dataset (normalizzazione, eliminazione di attributi e preprocessing)

e nella scelta degli algoritmi di apprendimento poi usati nel modello per la predizione, con conseguenti risultati differenti. Il lavoro è stato così suddiviso tra noi 4 componenti del gruppo: Spadoni e Rossi si sono occupati della gestione del dataset andando ad importarlo, modellarlo rimuovendo features non necessarie, ed effettuando un pre-processing dei dati, sia per la fase di analisi, sia per quella di predizione. Cannas e Abruzzese si sono occupati invece delle fasi di EDA (Exploratory Data Analysis), visualizzazione grafica dei dati e di predizione, andando a sviluppare i vari modelli poi testati. Per la stesura di questo report, ognuno di noi si è preso carico di una delle 4 sezioni da stilare. Gli strumenti utilizzati per abilitare la collaborazione sono stati:

- Google Colab per la parte di stesura del codice e per lo sfruttamento di rilevante potenza di calcolo virtuale, specialmente lato GPU;
- Google Docs per la stesura del presente report, vista la possibilità di poter lavorare in contemporanea, garantendo consistenza alle modifiche apportate.

2 Metodo proposto

2.1 Descrizione del dataset

Prima di analizzare il metodo proposto viene fatta innanzitutto una descrizione delle misurazioni effettuate dai sensori e immagazzinate nel dataset. Poiché i dataset di manutenzione predittiva reali sono generalmente difficili da ottenere e soprattutto da pubblicare, viene preso in considerazione un dataset sintetico e valido che riflette la reale manutenzione predittiva riscontrata nel settore.
Il dataset in questione è costituito da 10.000 record memorizzati in righe, ognuno dei quali con 14 attributi sistemati in colonne:

- **UID**: identificatore univoco che va da 1 a 10.000;
- **productID**: composto da una lettera L, M o H, per le varianti di qualità del prodotto (bassa (L, 50% di tutti i prodotti), media (M, 30%) e alta (H, 20%)) e da un numero di serie specifico della variante;
- **Type**: il tipo del prodotto realizzato. Coincide con la lettera presente nel productID;
- **temperatura dell'aria [K]**: generata utilizzando un processo di random walk successivamente normalizzato con una deviazione standard di 2K intorno a 300K;
- **temperatura di processo [K]**: generata utilizzando un processo di random walk normalizzato con una deviazione standard di 1K, aggiunta alla temperatura dell'aria più 10K;
- **velocità di rotazione [rpm]**: calcolata a partire da una potenza di 2860 W, sovrapposta a un rumore normalmente distribuito;
- **coppia [Nm]**: i valori di coppia sono normalmente distribuiti intorno a 40 Nm con un Ïƒ = 10 Nm e nessun valore negativo;
- **usura utensile [min]**: le varianti di qualità H/M/L aggiungono 5/3/2 minuti di usura all'utensile utilizzato nel processo;
- **Target**: valore che indica se un pezzo è guasto o meno;
- **Tipo di fallimento**: valore che indica il tipo di guasto, se presente;

Il guasto della macchina può avvenire in cinque modalità indipendenti:

- Guasto da usura dell'utensile (**TWF**);
- Guasto per dissipazione del calore (**HDF**);
- Interruzione di corrente (**PWF**);
- Rottura per sovratensione (**OSF**);
- Guasti casuali (**RNF**);

Se almeno una delle modalità di errore descritte sopra viene rilevata, il processo fallisce e l'etichetta "*Target*" è impostata ad 1. Per un'analisi migliore del dataset, sono state rimosse le colonne relative agli attributi "*UDI*" e "*productID*", poiché non contenevano informazioni utili al fine del problema. Dopo aver controllato che non ci fossero dati incompleti, si è proseguito con un'analisi esplorativa dei dati.

2.2 EDA (Exploratory Data Analysis)

L'EDA è un approccio per eseguire indagini iniziali sui dati per scoprire modelli, individuare anomalie, testare e verificare ipotesi con l'aiuto di statistiche e rappresentazioni grafiche.
Come prima cosa è stato utilizzato il metodo "`groupby`", che consente di suddividere i dati in gruppi basati su colonne/condizioni, di applicare funzioni/trasformazioni ai gruppi e combinare i risultati in un output. In questo caso, l'output mostra il **tipo di guasto** e la **relativa frequenza assoluta**. Ad una rapida occhiata della tabella sottostante, si nota subito come il numero di fallimenti rilevati risulti essere estremamente basso rispetto ai casi etichettati come "No *Failure*".

Target	Failure Type	count
0	No Failure	9643
	Random Failures	18
1	Heat Dissipation Failure	112
	No Failure	9
	Overstrain Failure	78
	Power Failure	95
	Tool Wear Failure	45

Tabella risultante group by

Successivamente, vengono calcolate: la **mediana relativa ad ogni tipologia di fallimento**, in modo tale da comprendere quali sono i valori mediani nel quale ricadono i fallimenti e la **mediana in relazione alla qualità del prodotto** (variabile "*Type*"). Di seguito vengono riportate le tabelle contenenti i risultati ottenuti.

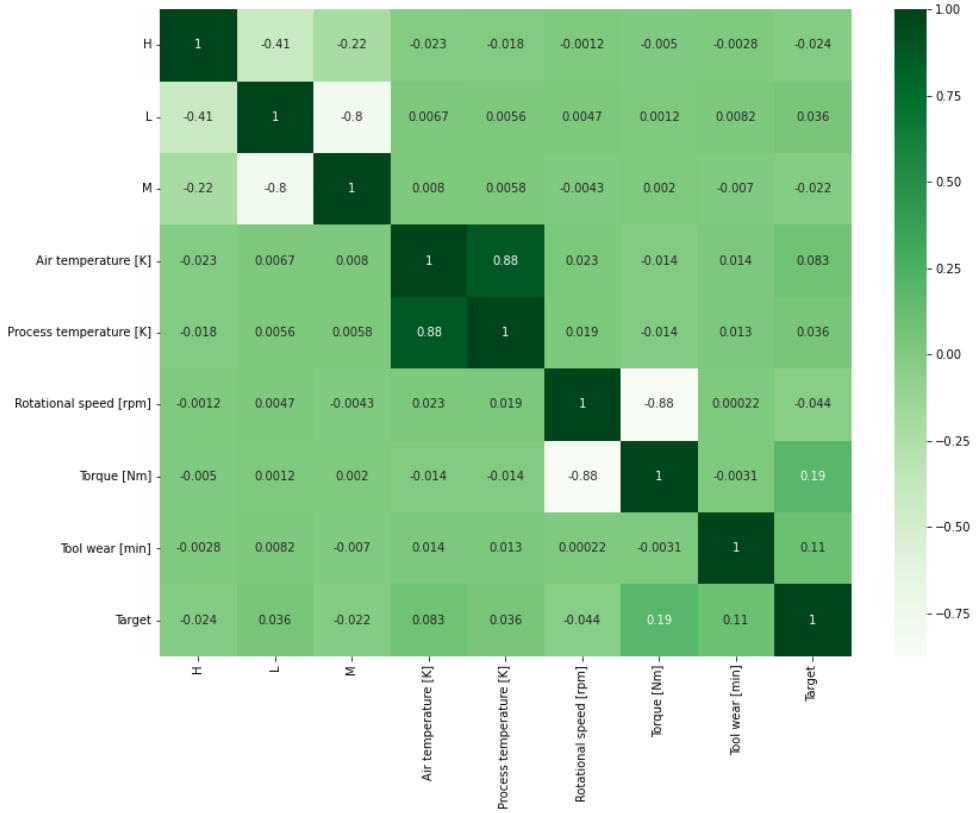
Target	Failure Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
0	No Failure	300.00	310.0	1507.0	39.80	107.0
	Random Failures	300.75	311.1	1490.0	44.60	142.0
1	Heat Dissipation Failure	302.45	310.7	1346.0	52.35	106.0
	No Failure	300.50	309.9	1438.0	45.20	119.0
	Overstrain Failure	299.45	310.1	1362.5	56.75	207.0
	Power Failure	300.40	310.2	1386.0	63.60	100.0
	Tool Wear Failure	300.40	310.3	1521.0	37.70	215.0

Tabella mediana tipologia fallimento

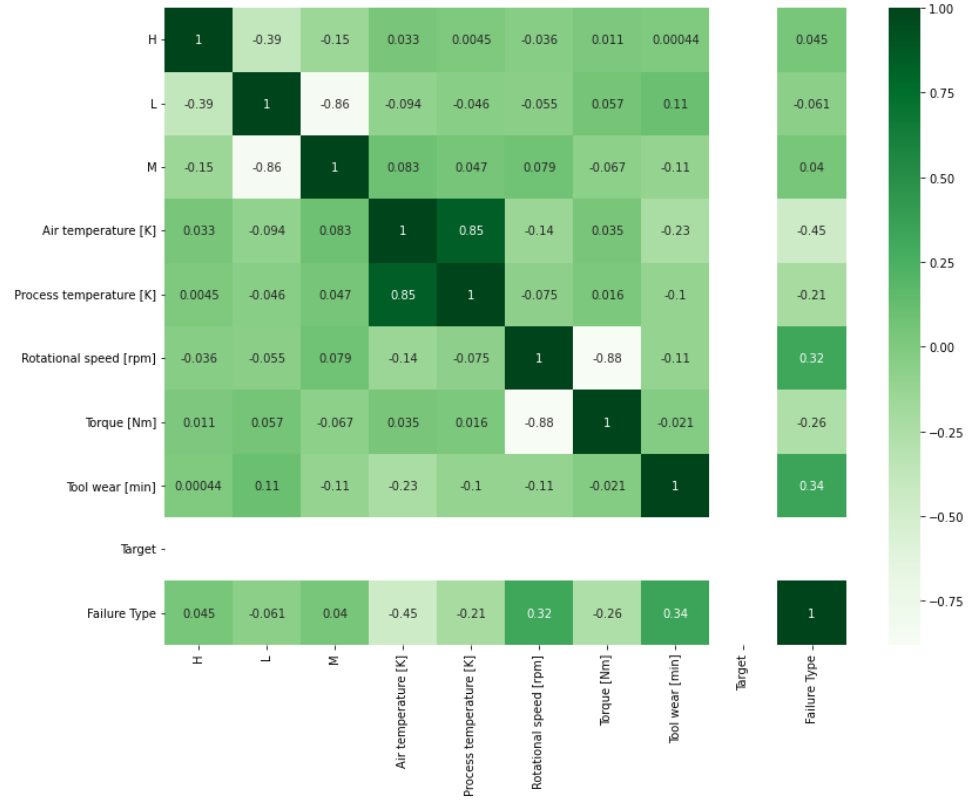
Type	Target	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
H	0	299.7	309.9	1502.0	40.2	106.0
	1	302.0	310.2	1371.0	53.8	147.0
L	0	300.1	310.1	1508.0	39.7	107.0
	1	301.2	310.4	1362.0	53.9	182.0
M	0	300.1	310.0	1506.0	40.0	105.0
	1	302.0	310.6	1372.0	51.6	125.0

Tabella mediana tipologia prodotto

Dopo aver trovato le due mediane, sono stati codificati il tipo di macchinario e il tipo di guasto in un unico data frame, per poter calcolare le matrici di correlazione. Tra quelle che sono le proprietà intrinseche dei dati, si può osservare la presenza di alcune forti correlazioni tra le varie coppie di attributi: ciò non sorprende affatto in quanto questi dati, seppur ottenuti in maniera casuale, sono legati matematicamente. Sono state calcolate e riportate in seguito due matrici; all'interno della prima possiamo osservare la correlazione che vi è tra le varie features, mentre nella seconda la correlazione tra i vari tipi di fallimento. Analizzandole si può notare ad esempio una significativa correlazione positiva pari a 0.88 tra la temperatura dell'aria e quella di processo, mentre esiste una significativa correlazione negativa pari a -0.88 tra la velocità di rotazione e il momento torcente.



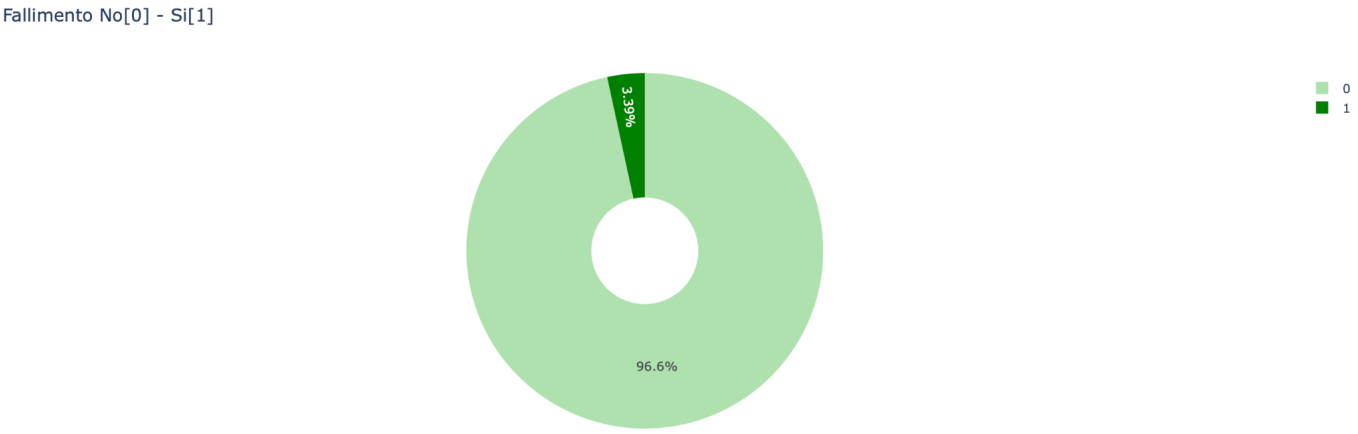
Matrice di correlazione tra le features



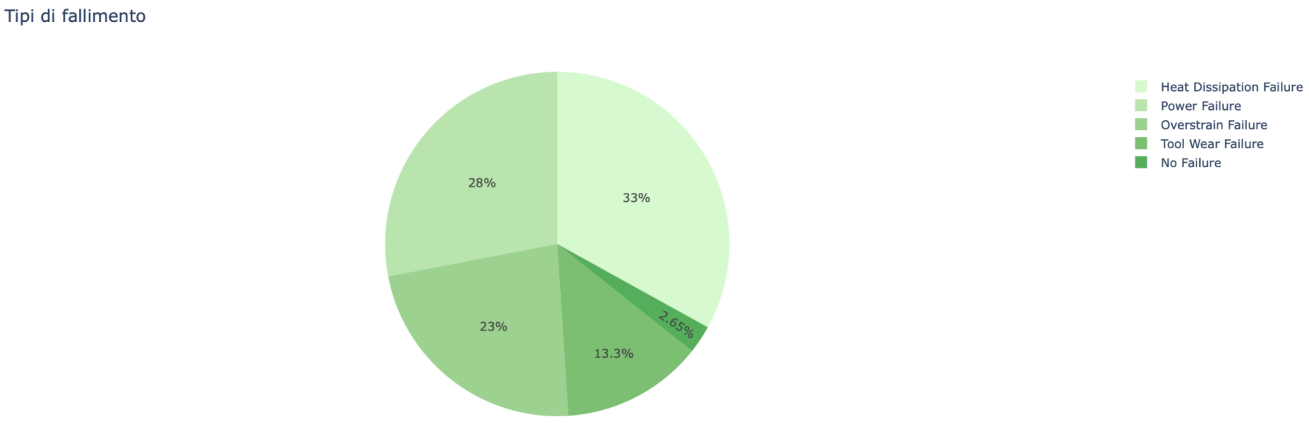
Matrice di correlazione dei fallimenti delle features

2.3 Visualizzazione dei dati

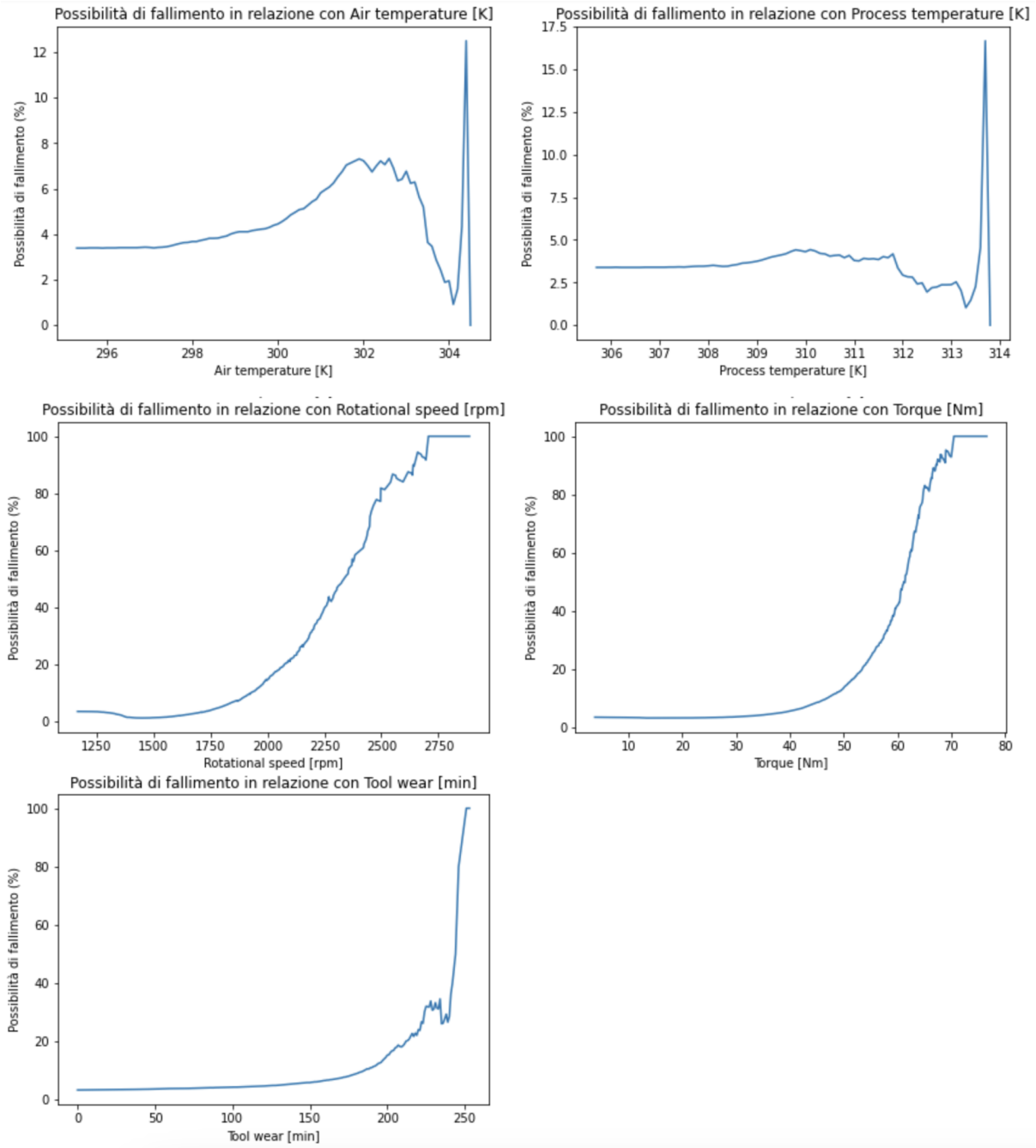
Durante la fase di visualizzazione dei dati, è stato riscontrato uno **sbilanciamento** del dataset, in quanto il numero di fallimenti dei macchinari è risultato essere pari al 3.39%. Ciò si può osservare dal grafico a torta riportato sotto, rappresentante la percentuale di fallimento (**fallimento si-no**):



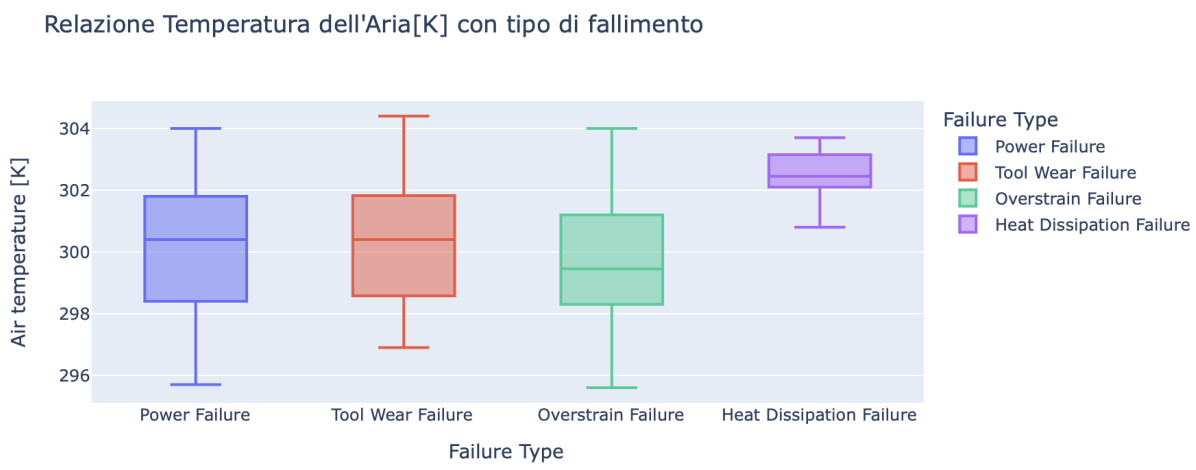
Per analizzare più in dettaglio le tipologie di fallimento tra quelle riscontrate, sono state evidenziate le percentuali per ogni **tipo di fallimento** rispetto al totale:



Successivamente il focus dell'analisi è stato spostato sulle probabilità di fallimento collegate alle varie features, per osservare intorno a quali valori si verificano fallimenti:



Prendendo ad esempio in considerazione il primo grafico, il quale denota la probabilità di fallimento in base alla temperatura dell'aria, si evince che quando quest'ultima supera i 304K, il 12% degli input falliscono, mentre per valori inferiori la probabilità di fallimento si aggira tra il 3% e l'8%.
Oltre al calcolo della probabilità è stata visualizzata anche la relazione tra ogni feature e il tipo di fallimento, osservando per quali valori si verificano i vari tipi di guasti:



Come mostrato dalla figura riportata sopra, si può notare che per valori compresi tra 298K e 302K, si verificano i seguenti fallimenti:

- Power Failure
- Tool Wear Failure
- Overstrain Failure

Mentre per valori compresi tra 302K e 304K si verifica come unico fallimento "Heat Dissipation Failure".
Per ogni tipologia di fallimento, è stata cercata una correlazione con le relative varianti di qualità del prodotto (H-qualità alta, L-qualità bassa, M-qualità media). Come presupposto, si può vedere che con una qualità di tipo "H" si hanno il minor numero di fallimenti, al contrario di "L", nel quale si hanno i valori maggiori.

Type	H	L	M
Failure Type			
Heat Dissipation Failure	8	74	30
No Failure	979	5757	2916
Overstrain Failure	1	73	4
Power Failure	5	59	31
Random Failures	4	12	2
Tool Wear Failure	6	25	14

Tabella per la relazione tra tipo di fallimento e qualità del prodotto

2.4 Data preprocessing

Encoding

All'interno della fase di encoding (codifica), sono state convertite le etichette in un formato numerico adeguato per i modelli poi utilizzati. Per fare ciò sui dati a disposizione, è stata utilizzata la funzione "fit", la quale esprime quali valori sono univoci, assegnando ad essi un valore e restituendo così le etichette codificate.

"Un algoritmo di apprendimento automatico deve essere in grado di comprendere i dati che riceve. Nel nostro caso, categorie come "H", "M" e "L" devono essere convertite in numeri. Per risolverlo, possiamo ad esempio convertirli in etichette numeriche con "1" per L, "2" per M e "3" per H."

Split test e train

All'interno di questa fase il dataset viene splittato in dati di testing e dati di training. La divisione test-train permette di stimare le prestazioni degli algoritmi di apprendimento tramite i dati di addestramento, convalidandoli con quelli di test.

"Un **training set** è un insieme di *esempi* ad ognuno dei quali è associata una risposta, il valore di un attributo-goal, ossia un valore categorico, cioè una classe, o un valore numerico (nel nostro caso è associato il valore del fallimento dell'input). Tali esempi vengono utilizzati per addestrare un modello predittivo supervisionato capace di determinare l'attributo-goal per nuovi esempi. Un modello addestrato può essere valutato su un nuovo insieme di esempi, il **test set**, non utilizzati in fase di addestramento."

Fonte: https://it.wikipedia.org/wiki/Training_e_test_set

Normalizzazione

Durante questa fase è stata "normalizzata" la gamma di variabili indipendenti e le features dei dati, per poter ridurre il rumore e renderle dunque più vicine tra loro. Questa fase è importante per gli algoritmi di apprendimento che calcolano le distanze tra i dati, per esempio il KNN.

"Molti classificatori calcolano la distanza tra due punti in base alla distanza euclidea. Se una delle caratteristiche ha un'ampia gamma di valori, la distanza sarà regolata da questa particolare caratteristica. Pertanto, l'intervallo di tutte le caratteristiche dovrebbe essere normalizzato in modo che ciascuna caratteristica contribuisca in modo approssimativamente proporzionale alla distanza finale."

Fonte: https://en.wikipedia.org/wiki/Feature_scaling

2.5 Sviluppo del modello

I modelli che sono stati utilizzati, sono stati inseriti all'interno di una lista, così come le rispettive istanze e sono:

- Regressione logistica
- KNN
- Support Vector Machine
- Random Forest
- Ada Boost
- XG Boost
- Naive Bayes
- Decision Tree Classifier
- Multi Layer Perceptron

Dopo aver aggiunto i classificatori e le istanze alle rispettive liste, è stato creato un data frame da ciò:

	Classificatori	Istanze
0	Regressione Logistica	regLog
1	K-Nearest Neighbour	knn
2	Support Vector Machine	supVectMac
3	Random Forest	randF
4	Ada Boost	ada
5	XG Boost	xgb
6	Naive Bayes	nb
7	Decision Tree Classifier	dtc
8	Multi Layer Perceptron	mlp

Tabella dei classificatori e delle istanze

- **Regressione Logistica:**
  La regressione logistica è un algoritmo di apprendimento supervisionato che studia la relazione che c'è tra una variabile dipendente e una indipendente, producendo dei risultati in formato binario utilizzati in

seguito per poter prevedere l'esito di una variabile categorica (in questo caso fallimento o meno).

- **KNN:**
  Il KNN è un algoritmo di apprendimento automatico supervisionato, usato per la risoluzione di problemi di classificazione. Memorizza tutti i casi disponibili e classifica i nuovi in base alla somiglianza; l'oggetto viene quindi classificato in base ai suoi vicini (identificando con "K" il numero di vicini). In questo caso si avrà una divisione tra input che falliscono e input che non falliscono, con i nuovi input che somiglieranno a uno dei due gruppi precedenti.

- **Support Vector Machine (SVM):**
  Il Support Vector Machine (SVM) è un algoritmo di apprendimento automatico supervisionato usato sia per la classificazione che regressione. Il suo funzionamento consiste nel creare una linea o un iperpiano per la separazione dei dati in classi. In questo caso le classi sono fallimento/non fallimento.

- **Random Forest:**
  Il Random Forest è un insieme di alberi di decisione che vengono utilizzati per raggiungere un unico risultato. L'algoritmo permette di eseguire sia la regressione che la classificazione, offrendo maggiore stabilità e precisione.

- **ADA Boost:**
  L'ADA Boost è un algoritmo che formula un numero H di ipotesi tramite l'algoritmo di "ensemble boosting" a partire da un training set di N esempi. Ogni ipotesi è diversa dalle altre, poiché costruita con parametri differenti e per ogni ipotesi, viene assegnato un peso per misurare la sua efficacia. L'algoritmo alla fine elabora l'ipotesi finale a maggioranza pesata.

- **XGBoost:**
  XGBoost è un algoritmo di machine learning che si basa su un insieme di algoritmi di apprendimento ad albero di decisione. È un algoritmo che permette di risolvere un'ampia gamma di problemi di previsione, classificazione e regressione.

- **Naive Bayes:**
  Naive Bayes è un algoritmo utilizzato per poter risolvere problemi di classificazione e apprendimento automatico utilizzando il teorema di Bayes. Questo teorema permette di andare a calcolare, per ogni istanza, la probabilità di appartenenza ad una data classe. Nello specifico Gaussian Naive Bayes assume che ogni parametro (feature) ha una capacità indipendente di predizione della variabile di output. La combinazione della predizione per tutti i parametri è la predizione finale, la quale indica la probabilità di una variabile dipendente di essere classificata in ogni gruppo. La classificazione finale è assegnata al gruppo con la probabilità maggiore.

- **Decision Tree Classifier:**
  Decision Tree è un algoritmo di apprendimento automatico usato per problemi di classificazione in cui i dati vengono continuamente suddivisi in base ad un parametro. L'albero è formato da due entità, cioè i nodi e le foglie.

- **Multi Layer Perceptron:**
  Il Multi Layer Perceptron è un modello di rete neurale artificiale che mappa un insieme di dati in input in un insieme di output appropriato. È costituito da strati multipli di nodi, in cui ogni strato è connesso al successivo con una funzione di attivazione lineare. Esso utilizza una tecnica di apprendimento supervisionato che prende il nome di "backpropagation" per poter allenare la rete e si differenzia dal percettrone lineare standard poiché può distinguere i dati che non sono separabili linearmente.

Modelling

È stata creata una classe "Modellazione" per poter utilizzare i classificatori sul nostro dataset. Si inizia con la creazione di due liste: una contenente le accuratezze dei modelli e l'altra il tempo di esecuzione di ogni modello. Il modello viene adattato in base ai dati di addestramento forniti e successivamente viene avviato il tempo di esecuzione. Una volta lanciato si va a calcolare l'accuratezza di quel modello, aggiungendola alla lista e una volta terminata l'esecuzione dell'algoritmo viene bloccato il tempo di esecuzione.

Gestione risultati

Tutte le informazioni dei risultati vengono racchiuse all'interno di un data frame che prende in input il modello utilizzato, la sua accuratezza e il tempo di esecuzione. Questi dati vengono poi passati in una fase successiva ad altre funzioni che permettono di trovare il miglior modello per accuratezza e tempo di esecuzione.

Per il miglior modello è stata costruita una matrice di classificazione contenente i valori per la valutazione dell'accuratezza dell'algoritmo, quali la sua precision, recall, f1 score e support (oltre al mostrare la sua accuracy e il suo tempo di esecuzione). Infine sono state determinate le prestazioni del miglior modello di classificazione tramite la costruzione di una matrice di confusione.

3 Risultati Sperimentali

3.1 Istruzioni per la dimostrazione

Come già spiegato nell'introduzione di questo report, il lavoro è stato svolto utilizzando la piattaforma Google Colab, che oltre ad offrire un'ottima potenza di calcolo virtuale aggiuntiva, permette di lavorare contemporaneamente con altri utenti tramite un link condiviso, ma soprattutto di poter scrivere ed eseguire il codice a sezioni. Ciò è molto utile per analizzare più precisamente le varie parti del codice e i relativi output, tenendo traccia man mano dell'andamento del notebook. Inoltre, è possibile andare a modificare e controllare i risultati delle modifiche senza dover necessariamente rieseguire tutto il codice. Per poter eseguire correttamente lo script contenente il qui presente lavoro, è necessario seguire questi passaggi:

- Aprire un nuovo notebook Colab e salvarlo con l'estensione `.py`;
- Inserire le porzioni di codice cella per cella;
- Prima di eseguire lo script, creare all'interno del file manager di Colab una sottodirectory contenuta nella directory "`Content`" e rinominarla "`DataSet`", nella quale andare ad inserire il file `.csv` contenente il dataset (`predictive_maintenance.csv`);
- Infine far partire il runtime, scegliendo se eseguire le celle tutte insieme oppure una alla volta.

3.2 Comparazione tra le diverse configurazioni e soluzioni

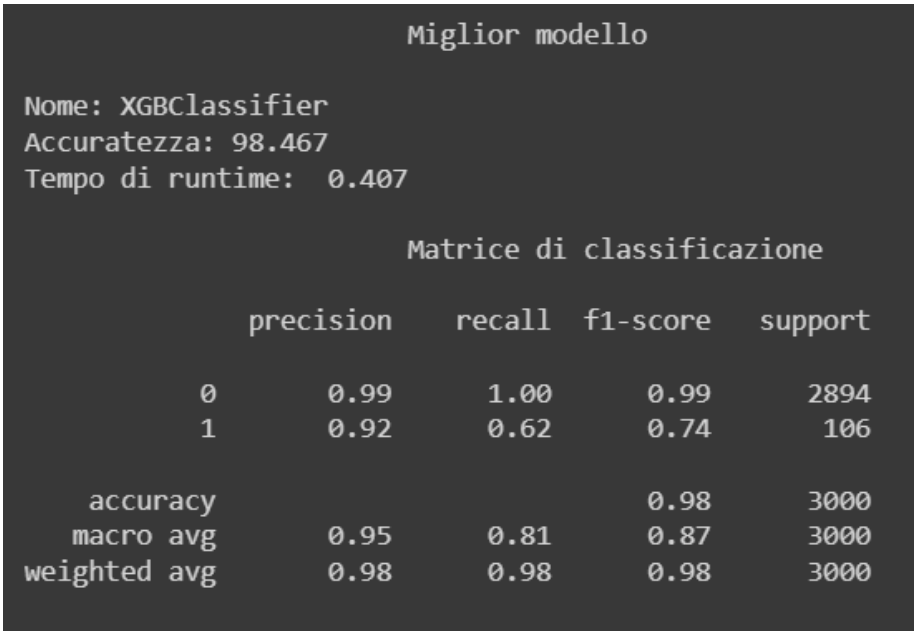
Come già detto il modello è stato testato svariate volte con diversi algoritmi di ML, quali Regressione logistica, K-Nearest Neighbors, Support Vector Machine, Random Forest, Ada Boost, XGBoost, Naive Bayes, Decision Tree Classifier e Multi Layer Perceptron. Si è scelto di utilizzarne un numero cospicuo per avere un'idea più chiara su quali fossero gli algoritmi più (e meno) performanti e per esplorare soluzioni più variegate e differenti, utilizzando appunto algoritmi di base piuttosto diversi tra di loro.
I metri di giudizio per l'ordinamento usati sono stati l'accuratezza e il tempo di esecuzione, anche se quest'ultimo risulta essere abbastanza ininfluente per il ranking degli stessi. Come si può vedere dall'immagine sottostante, tutti gli

algoritmi applicati al modello designato risultano avere un'accuracy più che notevole, essendo per tutti compresa tra il 96% e il 98%. Ciò che invece varia notevolmente è il tempo di esecuzione, visto che si va dall'ordine dei millesimi di secondo per Naive Bayes ai quasi 10 secondi per il Multi Layer Perceptron.

	Modelli	Accuratezza	Tempo di esecuzione (s)
0	XGBClassifier	98.467	0.399600
1	RandomForestClassifier	98.400	0.794466
2	MLPClassifier	97.900	9.634658
3	DecisionTreeClassifier	97.833	0.025432
4	KNeighborsClassifier	97.367	0.149063
5	AdaBoostClassifier	97.233	0.295955
6	SVC	97.200	0.257211
7	LogisticRegression	96.933	0.026256
8	GaussianNB	96.100	0.003678

Tabella per il ranking dei modelli basato su accuracy e tempo di esecuzione

In pressoché tutte le esecuzioni, l'algoritmo "*XGB Classifier*" restituiva un'accuratezza del 98% ed un tempo di esecuzione pari a 0,4/0,5 s. Per uno studio più approfondito, sono stati estratti e visualizzati in forma tabellare i dati di XGB relativi alla classificazione dei records sull'attributo target (*precision*, *recall*, *f-1 score* e *support*).



Valori miglior modello e matrice di classificazione

- ***precision***: la misura dell'esattezza del classificatore. É definita come il rapporto tra il numero dei positivi e la somma di positivi e falsi positivi. In altre parole, per tutte le istanze classificate positivamente, indica la percentuale delle classificazioni corrette;
- ***recall***: la misura della completezza del classificatore. É definita come il rapporto tra il numero dei positivi e la somma di positivi e falsi negativi. In altre parole, per tutte le istanze realmente positive, indica la percentuale di quelle classificate correttamente;
- ***f-1 score***: media armonica pesata di precision e recall, con un range possibile di valore che va un massimo di 1,0 e un minimo di 0,0.
- ***support***: il numero delle occorrenze effettive di ogni classe all'interno del dataset.

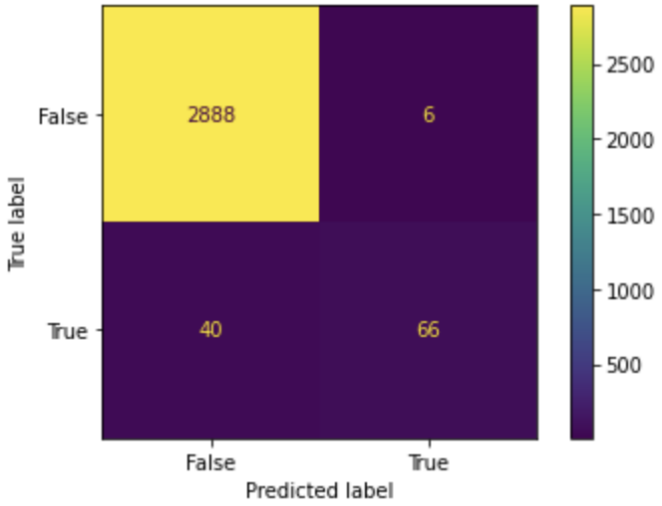
Osservando le varie soluzioni già esistenti online, si è notato come queste seguano tutte un approccio simile a quello adottato in questo lavoro e come, di conseguenza, anche i risultati ottenuti siano pressoché uguali. Ovviamente questi ultimi possono variare per i più diversi fattori, quali ad esempio un approccio totalmente differente al problema, una gestione del dataset diversa con una relativa fase di preprocessing più o meno accurata e l'utilizzo di ulteriori modelli rispetto a quelli applicati in questa configurazione. Detto ciò, l'algoritmo che risulta comunque essere migliore (quando usato), utilizzando sempre come

metrica di giudizio l'accuratezza, sembra essere nella maggior parte dei casi XGBoost. In alcune soluzioni proposte, invece, il "primo posto" se lo aggiudicano altri algoritmi quali "*Random Forest*" o "*Bagging Classifier*", seppur con valori di accuratezza che differiscono nell'ordine dei centesimi.

4 Conclusioni

4.1 Discussione dei risultati ottenuti

Dopo aver analizzato i risultati nel paragrafo precedente e dopo aver mostrato i risultati riportati a pag. 17, viene visualizzata una matrice di confusione correlata al miglior modello, rappresentando l'accuratezza di classificazione statistica. Ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui il classificatore ha classificato la classe "vera" i come classe j. Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi, andando a trovare "veri positivi/falsi positivi" e "veri negativi/falsi negativi".



Matrice di confusione

- **Veri negativi** = 2888 (la predizione negativa è effettivamente negativa);
- **Veri positivi** = 66 (la predizione positiva è effettivamente positiva);
- **Falsi negativi** = 6 (la predizione negativa è effettivamente positiva);
- **Falsi positivi** = 40 (la predizione positiva è effettivamente negativa).

Da questi valori possiamo andare a calcolare varie metriche per la valutazione del modello migliore (la maggior parte delle quali già analizzate nel paragrafo 3.2). Di seguito viene computato il tasso di errore (Error Rate), il quale misura la

percentuale di errore delle previsioni sul totale delle istanze (varia da 0 (migliore) a 1 (peggiore)) applicando la seguente formula:

$$ERR = \frac{FP + FN}{TP + TN + FP + FN} = \frac{40 + 6}{66 + 2888 + 40 + 6} = \frac{46}{3000} = 0.0153 = 1.53\%$$

Dunque nel 1.53% dei casi il modello sbaglia predizione.

4.2 Limitazioni e lavori futuri

Una limitazione importante nel nostro lavoro riguarda il dataset. Come abbiamo descritto all'interno dell'introduzione, risulta difficile reperire dei dataset contenenti dati reali per questo ambito, perciò è stato utilizzato un dataset "artificiale" che può essere, come in questo caso, sbilanciato e che quindi porta ad avere delle previsioni non del tutto realistiche. A fronte del problema principale dello sbilanciamento, un lavoro futuro potrebbe essere quello di effettuare un'analisi dei dati su un dataset bilanciato, per poter ottenere dei risultati più attendibili.