Università degli Studi di Catania

Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Corso di Laurea in Ingegneria Informatica

Marco Andronaco

Realizzazione di Applicazioni di Manutenzione Predittiva tramite piattaforma Microsoft Azure

---

Relatore:
Chiar.mo Prof. Salvatore Cavalieri

Correlatore:
Salvatore Gambadoro

Anno Accademico 2020–2021

Sommario

La presente tesi tratta la realizzazione di applicazioni di Manutenzione Predittiva (PdM) attraverso la piattaforma *Microsoft Azure Machine Learning*, con fine ultimo la costruzione di modelli efficaci per la predizione dei guasti per due diverse applicazioni industriali.

Si approfondiscono metodi per preparare i dati per l'apprendimento come MICE (*Multiple Imputation by Chained Equations*), SMOTE (*Synthetic Minority OverSampling Technique*) e RUS (*Random UnderSampling*); inoltre, si accenna il funzionamento e si confrontano le performance di diversi modelli, quali *Logistic Regression*, *Decision Tree*, *Random Forest*, *Boosted Decision Tree*, *Support Vector Machine*, *Averaged Perceptron* e *Neural Network*, nei due scenari di manutenzione predittiva.

Si esaminano anche le principali metriche di valutazione dei modelli (*Accuracy*, *Precision*, *Recall*, *F-Score*, ROC, AUC) e si valuta quali siano le più importanti per gli ambiti in esame.

Si è voluto approfondire quest'aspetto dell'industria 4.0 vista la richiesta sempre più crescente di tecniche PdM in ambito industriale, nonchè i benefici economici e produttivi certamente non trascurabili.

Indice

1 Introduzione **1**

1.1 Industria 4.0 1

1.2 Strategie di manutenzione 1

1.3 Microsoft Azure 3

2 Componenti utilizzate **4**

2.1 MICE (Multiple Imputation by Chained Equations) 4

2.2 SMOTE (Synthetic Minority Oversampling TEchnique) . . . 8

2.3 RUS (Random UnderSampling) 9

3 Modelli **10**

3.1 Logistic Regression 10

3.2 Decision Tree 12

3.3 Random Forest 13

3.4 Boosted Decision Tree 15

3.5 Neural Network 17

3.6 Support Vector Machine 19

3.7 Averaged Perceptron 20

4 Metriche di valutazione dei modelli **21**

4.1 Accuracy 21

4.2 Precision, Recall, F-Score 22

4.3 Curva ROC (Receiver Operating Characteristic) 23

4.4 AUC (Area Under Curve) 23

4.5 Confusion Matrix 24

5 Sviluppo delle applicazioni **26**

5.1 Applicazione 1 26

5.2 Applicazione 2 30

6 Conclusioni **37**

7 Ringraziamenti **38**

1 Introduzione

1.1 Industria 4.0

Al giorno d'oggi si contano tre rivoluzioni industriali:

1. La nascita del motore a vapore, alla fine del 18esimo secolo,
2. L'inizio della produzione di massa, alla fine del 19esimo secolo,
3. L'uso delle tecnologie dell'informazione, alla fine del 20esimo secolo.

Oggi, all'inizio del 21esimo secolo, si può già parlare di "Industria 4.0" riferendosi all'utilizzo in ambito industriale della tecnologia dell'informazione e di moderni sistemi di comunicazione.[?][?]

Come detto prima, però, la tecnologia dell'informazione è parte integrante delle aziende già dalla terza rivoluzione industriale. Questa quarta rivoluzione è caratterizzata dalle infinite possibilità date dal coinvolgimento di Internet all'interno della catena produttiva, al fine di agevolare la raccolta e comunicazione di dati relativi all'utilizzo dei macchinari.

Al giorno d'oggi è possibile, grazie all'utilizzo di sensori IIoT (*Industrial Internet of Things*), elaborare grandi quantità di dati raccolti da diverse fonti, per poi effettuare studi e predizioni al fine di ottimizzare al meglio i processi industriali.

1.2 Strategie di manutenzione

In una delibera dell'OCSE del 1963, la manutenzione fu definita come la "funzione aziendale alla quale sono demandati il controllo costante degli impianti e l'insieme dei lavori di riparazione e revisione necessari ad assicurare il funzionamento regolare e il buono stato di conservazione degli impianti produttivi, dei servizi e delle attrezzature di stabilimento".

Tra la terza e quarta rivoluzione industriale si è passati a un approccio *proattivo*, che riesce a prevedere i problemi prima che questi accadano. Esistono infatti tre strategie di manutenzione:

Manutenzione reattiva: quando accade un guasto, si pensa a sistemare; questo comporta *downtime* e costi aggiuntivi.

Manutenzione preventiva: viene eseguita a intervalli regolari, per eliminare ogni rischio di guasto; la strategia funziona, ma è poco efficiente in termini di costo.

Manutenzione predittiva: vengono utilizzati algoritmi di Intelligenza Artificiale per elaborare i dati dei sensori e predire se un guasto sta per avvenire.[?]

È evidente che la prima strategia è la più *naïf*, e può portare a problemi: guastandosi all'improvviso, un macchinario potrebbe danneggiarne altri adiacenti; inoltre, la produzione rimane bloccata durante il fermo dato dal guasto, portando ulteriori danni economici.

La seconda strategia, la manutenzione preventiva, è usata negli ambiti in cui non sono ammessi guasti durante il periodo operazionale (ad esempio per i componenti degli aerei), ma costringe ad effettuare ricambi anche quando il componente non è usurato al 100%. Questo spreco in termini di costo è molto significativo ed ha senso cercare di limitarlo.

Infine, le tecniche di manutenzione predittiva permettono di sfruttare al 100% i macchinari prima di sostituirli, portando quindi costi minimi e una forte riduzione del *downtime*.

Questa tesi si propone di affrontare due scenari di questo tipo e trovare la soluzione migliore per ciascuno di essi utilizzando la piattaforma *Azure Machine Learning*.

1.3 Microsoft Azure

Azure è la piattaforma *cloud* di *Microsoft*, comprende più di 200 prodotti e servizi cloud mirati ad individui ed aziende, tra cui *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS).

Azure Machine Learning Studio è un pannello che offre uno spazio di lavoro completo per progetti di *machine learning*, semplificandone la creazione e l'ottimizzazione, grazie alla possibilità di manipolare dataset ed allenare modelli tramite un'interfaccia *drag&drop* immediata ed intuitiva. C'è anche la possibilità di aggiungere blocchi personalizzati al repertorio già presente. Questi blocchi eseguono codice *Python* o *R* e permettono di rimuovere ogni limite apparente del workflow a blocchi.

Attraverso la pagina *Designer* è possibile progettare una *pipeline* da mandare poi in esecuzione su una macchina virtuale in *cloud*. È possibile visualizzare passo dopo passo i risultati di ogni blocco, con possibilità di salvare i modelli allenati o effettuare il *deploy* sotto forma di servizio *cloud*.

2 Componenti utilizzate

2.1 MICE (Multiple Imputation by Chained Equations)

MICE è un metodo di "*multiple imputation*" (inserimento multiplo), usato per inserire i dati mancanti in un dataset sotto certe ipotesi riguardo ai motivi per cui questi dati sono mancanti; nello specifico, i dati devono essere "*Missing at Random*" e non "*Missing Completely at Random*". Il procedimento può essere spiegato così:

1. Si sostituisce ogni valore mancante con la media calcolata sulla sua rispettiva colonna.

2. Uno dei valori sostituiti torna ad essere indicato come mancante.

3. Si imposta il valore indicato allo step 2 come variabile dipendente e si applica un modello regressivo al resto del dataset.

4. Il valore mancante viene rimpiazzato con la predizione del modello regressivo.

5. Si torna allo step 2 e si ripete la procedura per ogni campo con dati mancanti finché ogni valore mancante non sarà rimpiazzato con la rispettiva predizione fatta dal modello regressivo associato. Ci si riferisce a questo intero procedimento con il nome di "ciclo".

6. Si ripetono gli step precedenti finché non si raggiunge un obiettivo di convergenza o si esegue un numero fisso di cicli.

Alla fine, i valori ottenuti vengono salvati per ottenere un dataset senza valori mancanti. A lungo andare, la distribuzione dei coefficienti del modello regressivo dovrebbe convergere e stabilizzarsi, eliminando dunque la dipendenza dall'ordine in cui i valori mancanti vengono inseriti.[?][?]

Esempio

Si supponga che una banca voglia predire il target demografico ideale a cui vendere la propria carta di credito.

Sono stati rimossi dei valori dal dataset iniziale; l'obiettivo è approssimarli usando un algoritmo MICE.

Ha senso togliere la colonna *purchased* poiché l'algoritmo si esegue sulle *feature* e non sulla *label*. Sostituendo i valori mancanti con la media della loro colonna si ottiene la tabella *dataset0*.

valori mancanti

age	experience	salary	purchased
25		50	0
27	3		1
29	5	110	1
31	7	140	0
33	9	170	1
	11	200	0

dataset0

age	experience	salary
25	7	50
27	3	134
29	5	110
31	7	140
33	9	170
29	11	200

Tuttavia, è poco plausibile che una persona con 7 anni di esperienza riceva un salario di 50.000$ mentre un'altra con 5 anni di esperienza ne riceve più del doppio. Questo accade perché i valori sono stati inseriti valutando la singola colonna, e ignorando tutto il resto del dataset.

A questo punto, si imposta come mancante il valore in *age* e si usano le restanti righe per allenare un modello di regressione lineare che ha come variabile dipendente la colonna *age*. La riga con il valore mancante sarà usata come *test set* e il valore ottenuto sarà quindi sostituito all'originale. Si fa lo stesso con i valori mancanti in *experience* e *salary*.

age regression

age	experience	salary
25	7	50
27	3	134
29	5	110
31	7	140
33	9	170
	11	200

experience regression

age	experience	salary
25		50
27	3	134
29	5	110
31	7	140
33	9	170
36.2532	11	200

salary regression

age	experience	salary
25	1.8538	50
27	3	
29	5	110
31	7	140
33	9	170
36.2532	11	200

dataset1

age	experience	salary
25	1.8538	50
27	3	72.7748
29	5	110
31	7	140
33	9	170
36.2532	11	200

A questo punto si salva il dataset ottenuto come "*dataset1*" e si calcola la matrice differenza tra gli ultimi due dataset, per poter valutare la convergenza del modello di regressione lineare. Si continua a generare nuovi dataset finché la matrice differenza tra gli ultimi due non sarà composta solo da valori inferiori ad una certa soglia, oppure per un numero fisso di cicli.

dataset0 - dataset1

age	experience	salary
0	-5.1462	0
0	0	-61.2252
0	0	0
0	0	0
0	0	0
7.2532	0	0

dataset2

age	experience	salary
25	0.9172	50
27	3	80.7385
29	5	110
31	7	140
33	9	170
34.8732	11	200

dataset2 - dataset1

age	experience	salary
0	0.9366	0
0	0	7.9637
0	0	0
0	0	0
0	0	0
1.38	0	0

dataset3

age	experience	salary
25	1.0015	50
27	3	79.9876
29	5	110
31	7	140
33	9	170
35.0019	11	200

dataset3 - dataset2

age	experience	salary
0	0.0842	0
0	0	0.751
0	0	0
0	0	0
0	0	0
0.1287	0	0

dataset4

age	experience	salary
25	0.9999	50
27	3	80.0007
29	5	110
31	7	140
33	9	170
34.9998	11	200

2.2 SMOTE (Synthetic Minority Oversampling TEchnique)

I macchinari industriali sono progettati per durare. Per questo motivo, negli scenari di manutenzione predittiva si ha quasi sempre a che fare con dataset fortemente sbilanciati, con molti casi negativi (funzionamento corretto) e di conseguenza pochi casi positivi (rotture).

Per bilanciare i dati, si usa una tecnica detta "SMOTE", che permette di creare nuovi campioni a favore della classe in minoranza, a partire dai pochi campioni già esistenti. L'algoritmo funziona nel seguente modo:

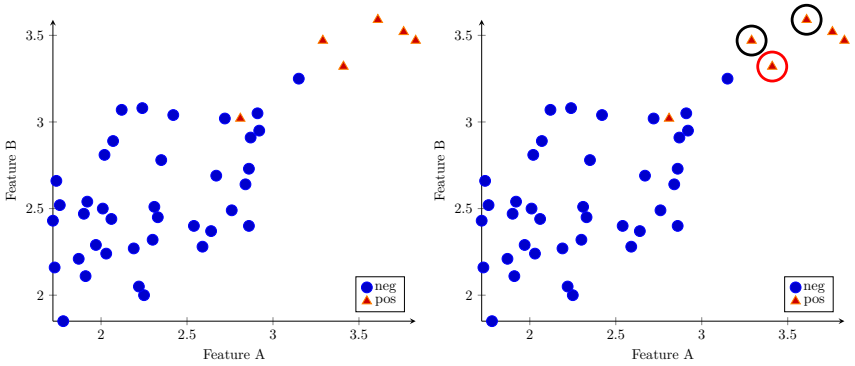
1. si sceglie un campione casuale dalla classe in minoranza,
2. si identificano i k campioni più vicini (con k configurabile),
3. si sceglie uno di questi vicini e si calcola il vettore che lo collega al campione scelto nello step 1,
4. si moltiplica il vettore per un numero casuale tra 0 e 1,
5. si somma il vettore risultante al campione scelto.

In altre parole, un campione viene scelto casualmente e spostato di poco in direzione di uno dei suoi k vicini.[?]

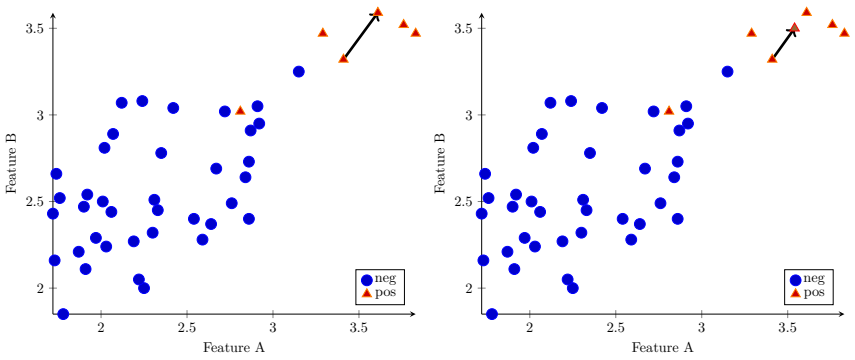
Esempio

Si supponga di avere un dataset composto da due *feature* e una *label*, fortemente sbilanciato, con molti dati appartenenti alla classe "*neg*" e pochi relativi alla classe "*pos*". Si applichi ora l'algoritmo SMOTE con $k = 2$.

Prima di tutto si sceglie un campione a caso dalla classe in minoranza e si identificano i 2 campioni più vicini. Sono stati cerchiati in rosso e nero rispettivamente il campione scelto e i due campioni ad esso più vicini.



Una volta scelto casualmente uno dei due campioni più vicini, si calcola il vettore che collega i due punti e si moltiplica quest'ultimo per un numero casuale compreso tra 0 e 1. Il risultato di questo prodotto sarà un nuovo campione da aggiungere al dataset.



L'algoritmo si ripete fino a raggiungere una percentuale di bilanciamento prefissata.

2.3 RUS (Random UnderSampling)

Un altro approccio per bilanciare i dati è il Random UnderSampling. L'algoritmo cerca di bilanciare i dati relativi a ciascuna classe eliminando campioni casuali scelti dalla classe in maggioranza. Anche qui, si fissa una percentuale da ottenere eliminando i campioni.[?][?]

3 Modelli

Una variabile si dice "dicotomica" se può assumere come unici valori 0 e 1 o riconducibili ad essi. Entrambi gli scenari di manutenzione predittiva affrontati in questa trattazione hanno previsto l'allenamento di modelli di classificazione binaria per variabili di questo tipo.

Si approfondiscono dunque i modelli di utilizzati sulle varie applicazioni.

3.1 Logistic Regression

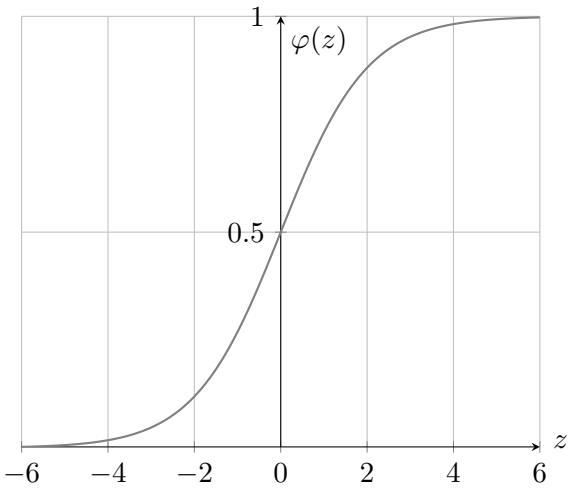
Il modello *logit*, noto anche come "modello logistico" o "regressione logistica", è un modello di regressione non lineare utilizzato quando la variabile dipendente è di tipo categorico. Tale modello, a partire da una variabile dicotomica, calcola la probabilità che essa acquisisca valore 1.[?][?]

Nella fase di addestramento, l'algoritmo di regressione logistica prende in input n campioni da un insieme di *training* (x). Ogni singolo campione è composto da m attributi e dalla classe corretta di appartenenza. L'algoritmo elabora una distribuzione di pesi (w) che permetta di classificare i campioni con le classi corrette. Poi calcola la combinazione lineare z del vettore dei pesi w e degli attributi x_m:

$$\begin{aligned} z &= \vec{x} \cdot \vec{w} + b \\ &= x_0 \cdot w_0 + x_1 \cdot w_1 + ... + x_m \cdot w_m + b \end{aligned} \tag{1}$$

La combinazione lineare z viene passata alla funzione logistica (*sigmoid*), che restituisce un numero compreso tra 0 e 1, ovvero la probabilità di appartenenza del campione alla classe 1 del modello.

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{2}$$



Infine, il risultato della funzione logistica viene convertito in un valore binario tramite una funzione di passo unitario (o quantizzatore) con una certa soglia, ad esempio:

$$\begin{cases} 1 \ se \ \varphi(z) > 0.5 \\ 0 \ \text{altrimenti} \end{cases} \tag{3}$$

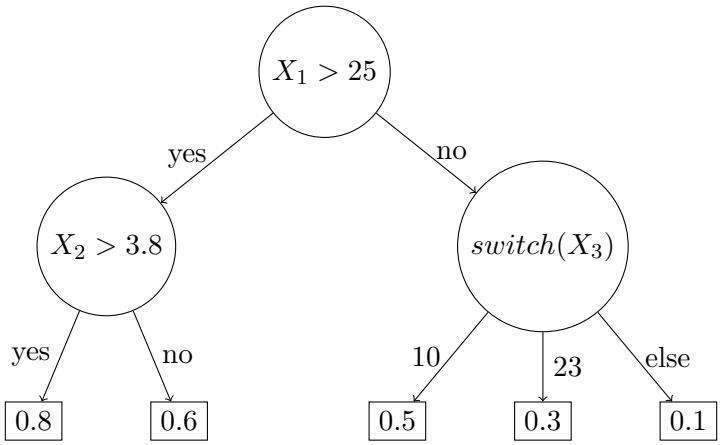
L'algoritmo di regressione logistica deve stimare il vettore dei pesi $\vec{w}$ ed il valore b in grado di minimizzare l'errore tramite la funzione statistica di massima verosimiglianza:

$$L_{w,b} = \prod_{i=1..N} \varphi(z_{(i)})^{y_i} (1 - \varphi(z_{(i)}))^{1-y_i} \tag{4}$$

3.2 Decision Tree

Un modello *Decision Tree* può essere visto come un insieme di regole non sovrapposte. Per esempio, si considerino tre variabili indipendenti X_1, X_2, X_3 e si rappresenti:

- con un cerchio, ogni regola del Decision Tree;
- con un rettangolo, ogni possibile risultato.



La procedura di creazione di un Decision Tree segue due step:

1. si costruisce un albero di decisione tramite allenamento supervisionato,
2. si effettua il cosiddetto "*pruning*", in italiano "potatura".

Il secondo passaggio è necessario poiché questo modello è molto sensibile all'*overfitting*: spesso succede di trovare alberi molto complessi e all'apparenza completi, ma troppo specifici in relazione ai dati di *training*.

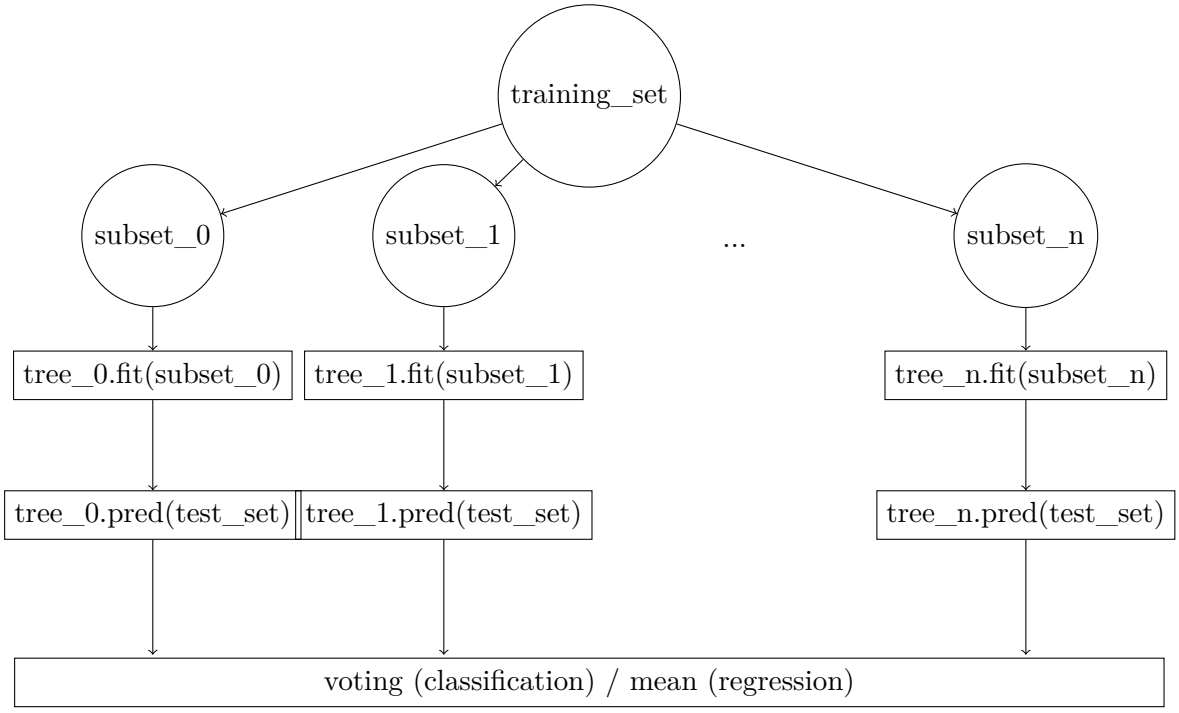
Si noti che ogni albero può avere un numero finito di foglie (risultati), dunque è impossibile usare questo modello per stimare valori continui. Per questo motivo, di solito, i modelli basati su Decision Tree si usano per risolvere problemi di classificazione, e non per predire il RUL (*Residual Useful Lifetime*) dei macchinari.[?]

3.3 Random Forest

Random Forest o *Decision Forest* è un un meta-modello, poiché lavora allenando diversi modelli e poi combinandoli tra loro, per ottenere una precisione maggiore rispetto a quella del singolo modello.

Un modello Random Forest funziona allenando un set di Decision Tree e può essere usato per fini di classificazione o regressione. Nel caso di classificazione, l'output è la classe scelta a votazione dal maggior numero di Decision Tree, mentre nel caso di regressione si calcola la media dei risultati degli alberi.

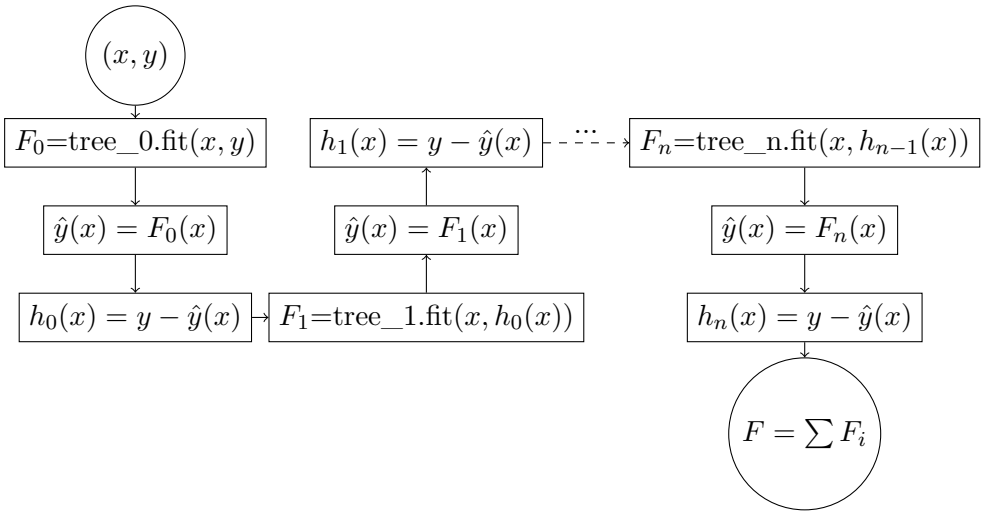
Ogni Decision Tree è generato a partire da sottoinsiemi casuali di *feature* e dati di *training*. Questa tecnica si chiama *bagging* (da *bootstrap aggregating*) ed aiuta a ridurre la varianza ed aumentare la robustezza rispetto al modello Decision Tree, scomponendo un grande problema in tanti più piccoli da risolvere in parallelo.[?][?]


training_set
subset_0
subset_1
...
subset_n
tree_0.fit(subset_0)
tree_1.fit(subset_1)
tree_n.fit(subset_n)
tree_0.pred(test_set)
tree_1.pred(test_set)
tree_n.pred(test_set)
voting (classification) / mean (regression)

3.4 Boosted Decision Tree

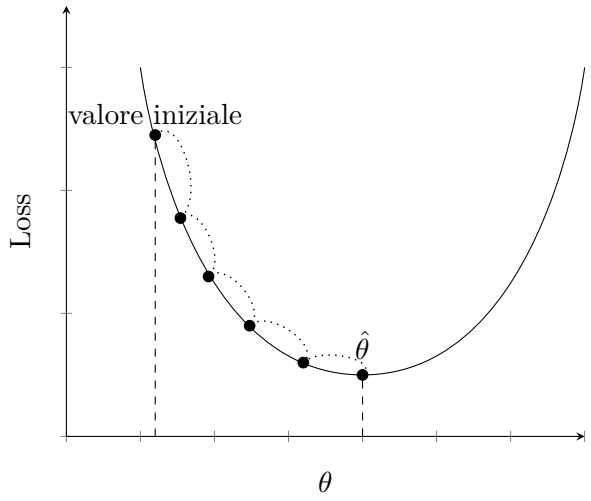
Al contrario del *bagging*, il *boosting* utilizza un processo totalmente sequenziale, con possibilità di scegliere un grado di errore soddisfacente a cui fermarsi. Il modello opera così:

1. Si allena un Decision Tree F sui dati iniziali (x, y).
2. Si applica $F(x)$ per predire $\hat{y}(x)$
3. Si calcolano i residui di questo Decision Tree: $h(x) = y - \hat{y}(x)$
4. Si torna allo step 1 con $y = h(x)$ finché non si è soddisfatti dell'errore.
5. Si elabora la predizione finale sommando le predizioni di tutti i Decision Tree accumulati.



Uno dei modelli di boosting più usati è il *Gradient Boosting*. Si chiama così perché usa un gradiente discendente per minimizzare una funzione legata ai residui, detta "*Loss Function*".

L'algoritmo, infatti, cerca di ottenere $\theta = 0$, dove θ è il gradiente della *Loss Function*, poichè minimizzare il gradiente significa trovare il punto in cui i residui sono minimi.



Alcune *Loss Function* sono *Mean Squared Error* e *Mean Absolute Error*:

$$\begin{aligned} \text{MSE:} \qquad & L(y, F(x)) = \frac{1}{N} \sum_{i=1}^{N} (y_i - F(x_i))^2 \\ \text{MAE:} \qquad & L(y, F(x)) = \sum_{i=1}^{N} |y_i - F(x_i)| \end{aligned} \tag{5}$$

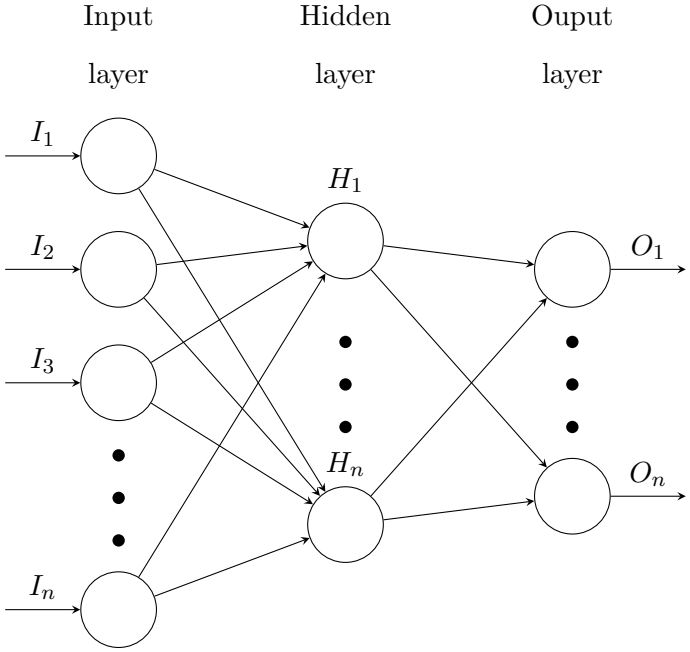
dove N è il numero di osservazioni.[?][?]

3.5 Neural Network

Una rete neurale è un modello computazionale non lineare costituito da un gruppo di interconnessioni di informazioni tramite "neuroni" artificiali; è un sistema adattivo che cambia la propria struttura in base a informazioni esterne o interne che scorrono attraverso la rete stessa durante la fase di apprendimento.

Ogni rete neurale riceve segnali esterni su uno strato di nodi di ingresso, ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato ai nodi successivi.

Questo è permesso dal fatto che una funzione $f(x)$ può essere definita come una composizione di altre funzioni $G(x)$, che possono a loro volta essere definite come composizione di altre funzioni. Il modello può essere rappresentato come una struttura a rete, dove le frecce rappresentano le dipendenze tra variabili, dette appunto "neuroni".



L'apprendimento, anche per le reti neurali, si ottiene minimizzando una funzione di costo (quindi annullandone il gradiente), la cui scelta dipende dal compito che la rete deve eseguire.

Normalmente una rete è formata da tre strati:

1. Nel primo abbiamo gli ingressi ("*Input*"): questo strato si preoccupa di trattare gli ingressi in modo da adeguarli alle richieste dei neuroni. Nel caso in cui i segnali in ingresso fossero già trattati può anche non esserci.

2. Il secondo strato è quello nascosto ("*Hidden*"), si preoccupa dell'elaborazione vera e propria e può essere composto anche da più colonne di neuroni.

3. Il terzo strato è quello di uscita ("*Output*") e si preoccupa di raccogliere e presentare i risultati dell'elaborazione.

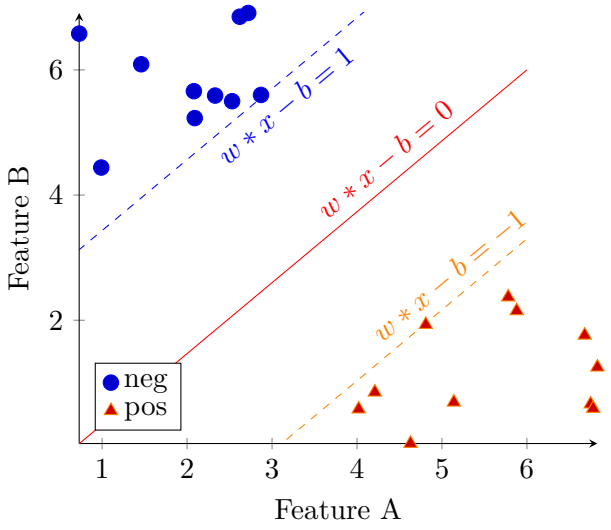
Queste reti possono essere anche molto complesse e coinvolgere migliaia di neuroni e decine di migliaia di connessioni. Lo strato *Hidden* può essere composto da N colonne, al fine di aumentare la complessità del modello.

Una particolarità di questo modello è l'impossibilità di spiegarne con la logica il funzionamento una volta allenato, poiché opera come una cosiddetta "scatola nera", di cui sono noti solo ingressi e uscite.[?][?]

3.6 Support Vector Machine

Le *support-vector machines*, o *support-vector networks* costituiscono un altro tipo di modelli per l'apprendimento supervisionato.

Dato un dataset di *training*, composto da n punti di forma $(x_1, y_1), ..., (x_n, y_n)$, dove le classi $y_i \in \{-1, 1\}$, il modello cerca il "*maximum-margin hyperplane*", ovvero l'iperpiano che divide il gruppo di punti x_i per cui $y_i = 1$ dal gruppo di punti per cui $y_i = -1$. Tale piano è definito in modo tale da massimizzare la distanza tra esso e gli x_i più vicini relativi ai due gruppi.[?]



Nel caso in cui i campioni non fossero linearmente separabili, l'algoritmo lavora su una dimensione aggiuntiva, calcolata in modo conveniente, per trovare una superficie curva che separi al meglio i campioni delle due classi.

3.7 Averaged Perceptron

Il percettrone è un modello lineare di classificazione binaria per apprendimento supervisionato; può essere visto come il caso più semplice di una Neural Network.

Esso consiste di un singolo nodo o neurone, che calcola la somma pesata degli input ed un *bias*. La somma pesata degli input del modello è chiamata "*activation*" ed è tale che:

$$\text{Activation} = \text{Weights} + \text{Inputs} + \text{Bias} \tag{6}$$

Le predizioni saranno:

$$\begin{cases} \hat{y} = 1 & \text{Activation} > 0.0 \\ \hat{y} = 0 & \text{Activation} \leq 0.0 \end{cases} \tag{7}$$

Anche qui, i coefficienti del modello (*weights*) vengono allenati usando l'algoritmo del gradiente discendente.

I campioni del dataset di *training* vengono presentati al modello uno alla volta; il modello fa una predizione e viene calcolato l'errore. I pesi del modello vengono quindi aggiornati per ridurre gli errori su quel campione. Questo processo si ripete per ogni campione nel dataset di *training* e viene chiamato "*epoch*".

I pesi continuano poi ad essere aggiornati nel corso di diversi epoch, finchè l'errore del modello non cade sotto una certa soglia, oppure una volta raggiunto un numero massimo di iterazioni.

Come SVM, il percettrone è un modello lineare, dunque il suo obiettivo è trovare un iperpiano in grado di separare i campioni delle due classi nello spazio delle *feature*.[?]

4 Metriche di valutazione dei modelli

Una volta allenato e sottoposto a un database di *testing*, un modello di classificazione binario può dare origine a quattro tipi di esiti:

True Positive: il modello ha previsto, correttamente, un caso positivo;

True Negative: il modello ha previsto, correttamente, un caso negativo;

False Positive: il modello ha previsto, erroneamente, un caso positivo;

False Negative: il modello ha previsto, erroneamente, un caso negativo.

Ha senso approfondire le diverse metriche che si utilizzano per confrontare i vari modelli, al fine di poter scegliere al meglio un classificatore adeguato alla manutenzione predittiva.

4.1 Accuracy

Si tratta della misura di performance più intuitiva, data dal ra nel caso di dataset bilanciati, che producono un numero di falsi positivi simile al numero di falsi negativi.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{8}$$

Come già detto, gli scenari di manutenzione predittiva presentano quasi sempre dataset fortemente sbilanciati, dunque si può dire che questa statistica sia in fin dei conti poco rilevante ai fini dell'ambito in questione.

4.2 Precision, Recall, F-Score

Si definisce "Precisione" o "*Precision*" il rapporto tra il numero di previsioni positive corrette ed il totale degli esiti positivi previsti dal modello. Un modello molto preciso è un modello che predice con sicurezza, senza falsi positivi, ma potrebbe non prevedere tutti gli eventi se non fosse abbastanza sensibile.

Si definisce "Sensibilità" o "*Recall*" il rapporto tra il numero di previsioni positive corrette ed il totale degli esiti positivi che si verificano effettivamente. Un modello molto sensibile è un modello che predice molti eventi, ma potrebbe non essere abbastanza preciso e dar luogo a falsi positivi.

Si definisce "*F-Score*" la media armonica delle metriche Precisione e Sensibilità.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{9}$$

$$\text{F-Score} \;=\; \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall+Precision}} \tag{10}$$

Nell'ambito della manutenzione predittiva, un falso negativo può avere un impatto nettamente maggiore rispetto a un falso positivo, dunque è utile scegliere modelli con elevata Sensibilità nel caso in cui l'obiettivo sia minimizzare i costi.

4.3 Curva ROC (Receiver Operating Characteristic)

Si definiscono "*True Positive Rate*" (TPR) e "*False Positive Rate*" (FPR):

$$\text{TPR} = \frac{\text{TP}}{\text{P}} \qquad \text{FPR} = \frac{\text{FP}}{\text{N}} \tag{11}$$

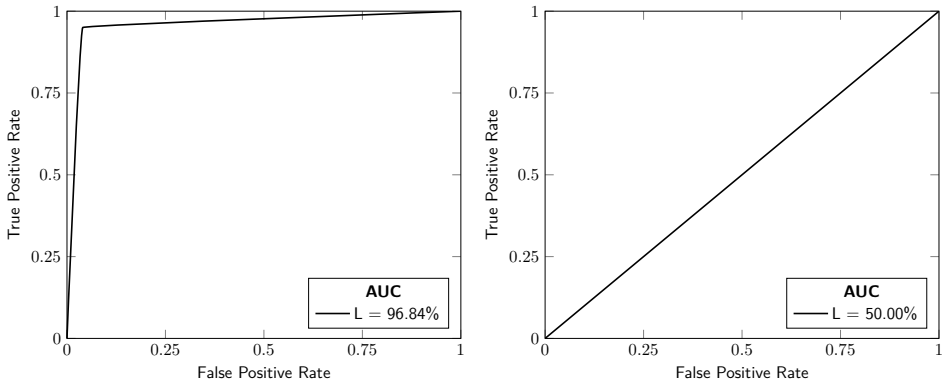
Dove P è il numero totale di campioni positivi ed N è il numero totale di campioni negativi.

ROC sta per "*Receiver Operating Characteristic*" ed è la curva 2D che lega TPR ed FPR al variare della soglia di classificazione da 0 a 1.[?]

In generale, più la curva ROC è vicina all'angolo superiore sinistro del grafico, più il modello predice in maniera efficace. Al contrario, un modello che predice in modo casuale avrà una curva ROC più vicina alla diagonale.

4.4 AUC (Area Under Curve)

Il parametro AUC indica la percentuale di area del grafico coperta dalla curva ROC, dunque più AUC è vicino a 1, più il modello in questione sarà preciso nelle sue osservazioni.



Questo parametro, a differenza dell'*Accuracy*, è sensibile allo sbilanciamento delle classi, ed è quindi più significativo per le applicazioni di manutenzione predittiva.[?][?]

4.5 Confusion Matrix

La matrice di confusione è uno strumento atto ad analizzare visivamente le performance di un modello di classificazione. Si tratta di una matrice che ha come righe le classi effettive e come colonne le classi previste dal modello.

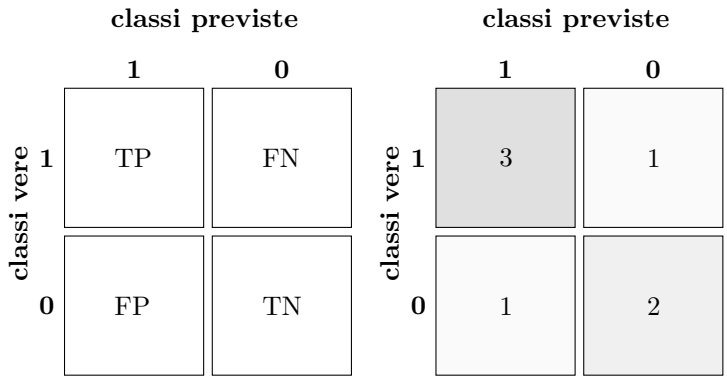
In generale, un modello è efficace se i valori lungo la diagonale principale della matrice sono nettamente più alti degli altri, poiché questi rappresentano i casi in cui il modello ha effettuato una predizione corretta.

Esempio

Si consideri la tabella già vista in precedenza nella sezione 2.1 e si supponga di aver allenato un modello a classificare la colonna *purchased*.

age	experience	salary	purchased	pred
25	1	50	0	0
27	3	80	1	0
29	5	110	1	1
31	7	140	0	0
33	9	170	1	1
35	11	200	0	1
37	12	210	1	1

La matrice di confusione relativa al modello sarà:


classi previste
1
0
classi vere
1
0
TP
FN
FP
TN
classi previste
1
0
classi vere
1
0
3
1
1
2

5 Sviluppo delle applicazioni

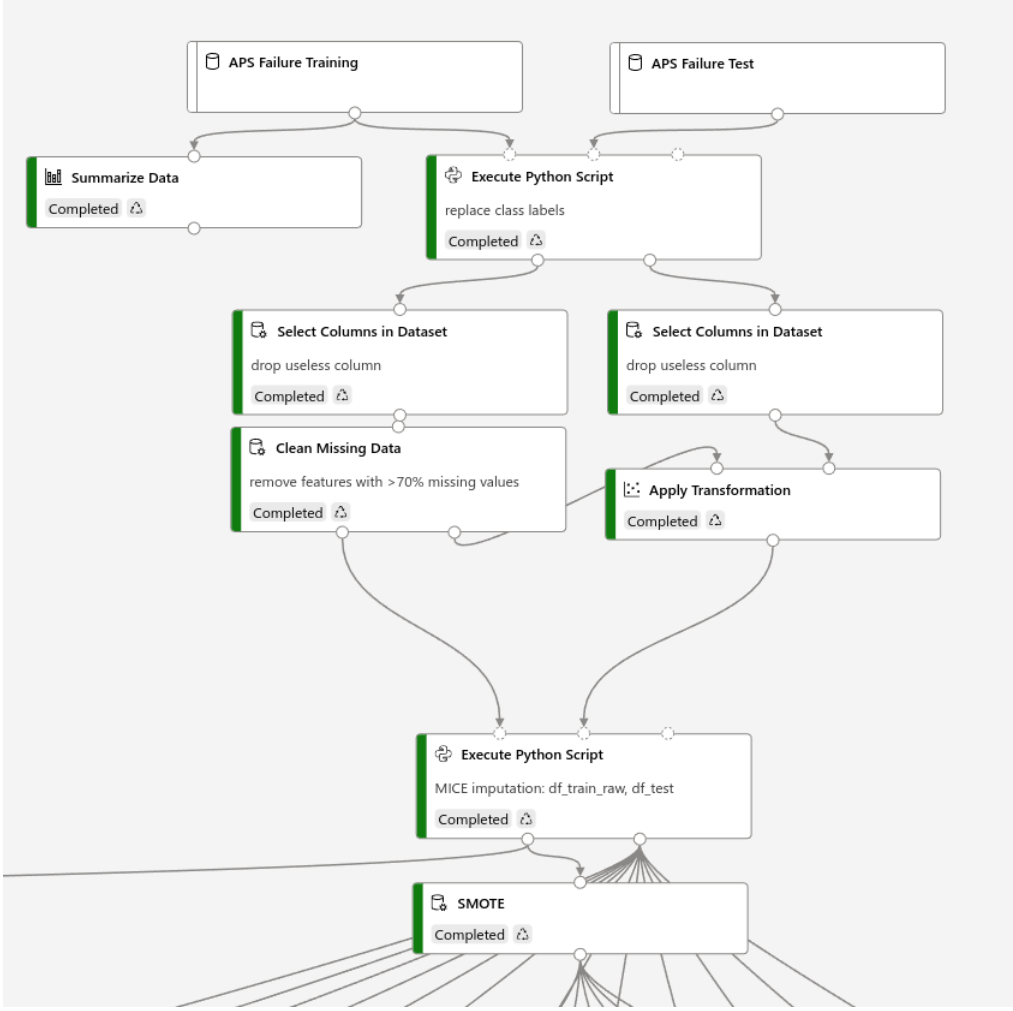
5.1 Applicazione 1

I veicoli superiori a 3.5 tonnellate utilizzano un impianto frenante ad aria compressa. A partire da un dataset fornito da SCANIA, si è formulato un modello ottimale di predizione dei guasti, che sia in grado di predire qualora un guasto sia dovuto o no ad un componente del sistema di frenata e ridurre al minimo i costi di classificazione, sapendo che i costi relativi a False Positive e False Negative sono rispettivamente 10 e 500.

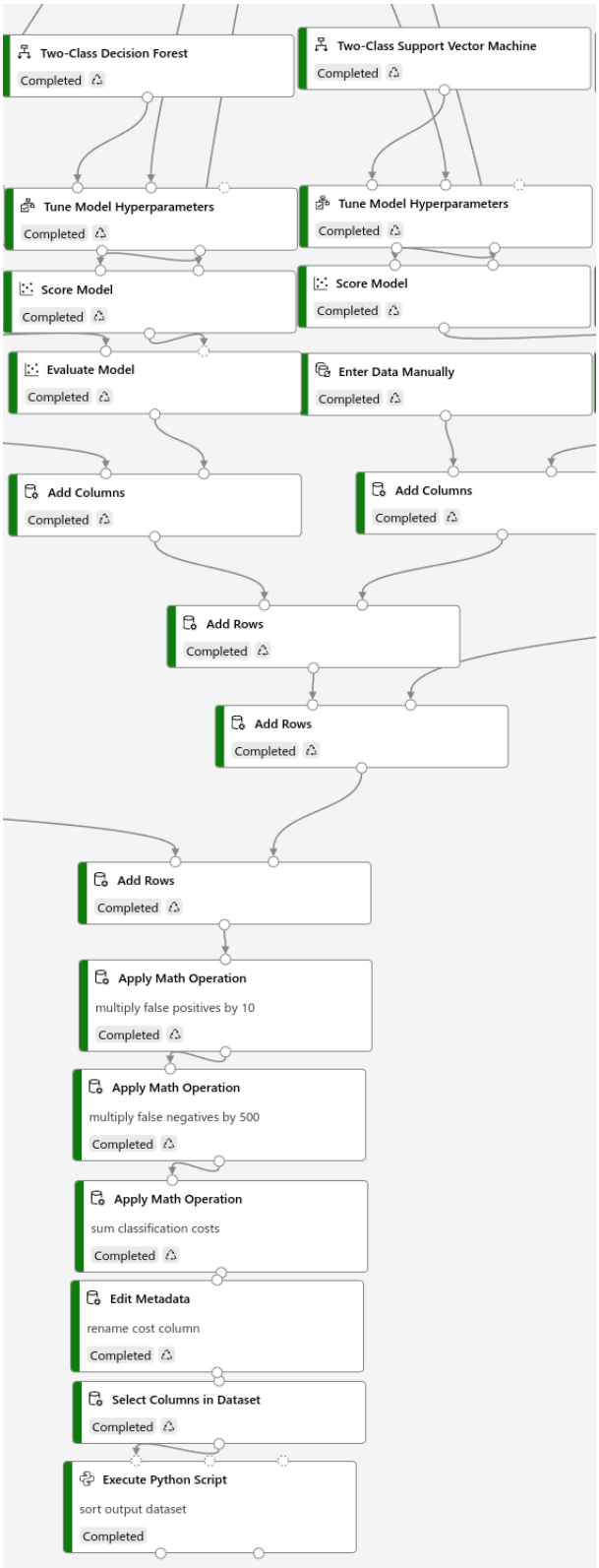
Pipeline

Una prima analisi del dataset rivela il suo problema principale: l'enorme numero di dati mancanti. Per risolverlo, sono state rimosse le colonne con più del 70% dei valori mancanti e si è usata una tecnica di inserimento MICE (sezione 2.1) con *Ridge Regression*[?] sul dataset. Questo algoritmo non è presente come blocco preimpostato di Azure, dunque è stato necessario l'utilizzo di un blocco "*Execute Python Script*" per implementarlo.

Una volta pulito, il dataset risulta fortemente sbilanciato; dunque è necessario un blocco SMOTE (sezione 2.2) che porti il dataset di *training* a una parità di casi positivi e negativi per quanto riguarda la *label* "*class*".



A questo punto il dataset è pronto per l'apprendimento e viene passato a diversi modelli per confrontarne le performance. Per ogni modello, è stato necessario un blocco "*Tune Model Hyperparameters*", che sceglie i parametri di apprendimento migliori al fine di massimizzare una certa metrica; in questo caso, si è scelta la metrica *Recall*, per aumentare la sensibilità del modello e quindi ridurre al minimo il numero di False Negative.


Two-Class Decision Forest
Completed
Two-Class Support Vector Machine
Completed
Tune Model Hyperparameters
Completed
Tune Model Hyperparameters
Completed
Score Model
Completed
Score Model
Completed
Evaluate Model
Completed
Enter Data Manually
Completed
Add Columns
Completed
Add Columns
Completed
Add Rows
Completed
Add Rows
Completed
Add Rows
Completed
Apply Math Operation
multiply false positives by 10
Completed
Apply Math Operation
multiply false negatives by 500
Completed
Apply Math Operation
sum classification costs
Completed
Edit Metadata
rename cost column
Completed
Select Columns in Dataset
Completed
Execute Python Script
sort output dataset
Completed

Risultati

È stato utile confrontare i risultati ottenuti in due casi:

1. Usando solo SMOTE,

2. Bilanciando con SMOTE fino al 30% per arrivare al 50% con RUS.

Model	SMOTE	RUS	Cost
Averaged Perceptron	30%	50%	13950
Neural Network	50%	-	17010
Support Vector Machine	50%	-	19900
Logistic Regression	50%	-	20200
Averaged Perceptron	50%	-	20380
Decision Forest	50%	-	22050
Support Vector Machine	30%	50%	22510
Neural Network	30%	50%	23440
Logistic Regression	30%	50%	23570
Decision Forest	30%	50%	24680
Boosted Decision Tree	30%	50%	31300
Boosted Decision Tree	50%	-	33310

Si evince facilmente che il modello con il costo di classificazione minore è Averaged Perceptron, allenato con SMOTE + RUS.

Un'ulteriore osservazione meno immediata è che Average Perceptron sia l'unico modello ad aver beneficiato di molto dall'utilizzo del Random UnderSampling. Tutti gli altri hanno ottenuto performance peggiori, fatta eccezione per il Boosted Decision Tree, che performa leggermente meglio con RUS sebbene sia l'algoritmo con il maggior costo e quindi il peggiore in questo caso.

Si può confrontare il risultato con altri precedentemente ottenuti in letteratura[?] sullo stesso dataset.

Model	Cost
Decision Forest	40570
k-Nearest Neighbor[?]	49850
Logistic Regression	61470
Support Vector Machine	545000

Dunque, con l'approccio di questa trattazione si è ottenuto un costo minimo pari a quasi 1/3 in meno rispetto ai risultati ottenuti in letteratura.

5.2 Applicazione 2

Si è mostrato quanto un approccio di manutenzione predittiva possa far risparmiare a un'azienda che finora ha effettuato solo manutenzione reattiva, facendo funzionare i macchinari fino alla rottura.

Partendo da un campione di dati riguardanti i guasti di 419 macchine in un periodo di due anni, si è cercato di ottenere un modello in grado di predire il guasto di ogni macchinario entro 90 giorni.

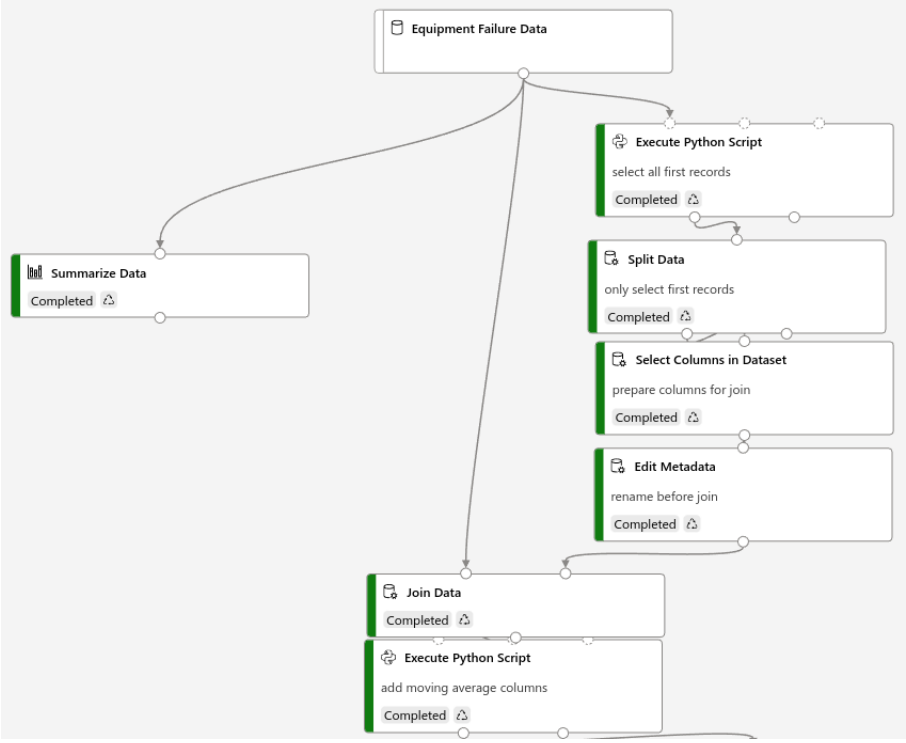
Scenario	Occorrenze	Costo	Costo Totale
Manutenzione superflua	9	1.500$	13.500$
Manutenzione opportuna	27	7.500$	202.500$
Guasto	383	30.000$	11.490.000$
Totale	419		11.706.000$

Pipeline

Il dataset è una serie temporale di rilevamenti effettuati ogni giorno nel corso di due anni. Per questo motivo, è stato utile creare diverse *lag feature*, ovvero

statistiche mobili calcolate su una certa finestra temporale antecedente al record in questione.

In particolare, è stata usata la funzione *shift()* della libreria *pandas* per generare indici mobili per media e deviazione standard di tutte le *feature* relative ai sensori, calcolati su una finestra pari a 28 giorni.



Successivamente, è stato necessario creare una classe da predire. Per farlo, è bastato aggiungere una colonna "*time_to_failure*", che associa a ogni record il numero di giorni restanti a quella macchina prima del prossimo guasto. Infine, è stata aggiunta la colonna *label* "*failure_target*", tale da valere 1 solo nei casi in cui il macchinario in questione fallisca entro 28 giorni.

Fatto questo, si è diviso il dataset in due diversi dataset per *training* e *testing*; il punto di *cut-off* è stato stabilito in base alla data, in modo che i due dataset coprano esattamente un anno ciascuno.

Prima di lanciare gli algoritmi di *training*, è stato opportuno rimuovere le colonne contenenti le date dal dataset di *training*, lasciando solo quelle relative ai dati dei sensori, le *lag feature* e le altre colonne categoriche.

Anche in questo caso è necessario effettuare un processo di SMOTE (sezione 2.2) sul dataset di *training*, essendo i dati fortemente sbilanciati verso la classe negativa.

Dalla tabella dei costi si può notare che un falso negativo costa molto più di un falso positivo, dunque anche per questa applicazione ha più senso impostare *Recall* come metrica da massimizzare nei blocchi *Tune Model Hyperparameters*. Questa strategia aumenterà la sensibilità del modello.

A questo punto si allenano i vari modelli da confrontare; in questo caso, è stato introdotto un modello aggiuntivo chiamato "*Dummy Classifier*". Questo modello predice solo 0, ovvero simula un approccio di manutenzione reattiva, in cui l'azienda lascia operare i macchinari fino alla rottura.

Una volta allenati i vari modelli, una nuova sfida è valutare quale sia quello più efficace dal punto di vista economico. Il blocco "*Score Model*" di Azure restituisce un dataset valutato, che è essenzialmente il dataset di *testing* con due nuove colonne:

"Scored Labels:" la classe prevista dal modello.

"Scored Probabilities": la probabilità che il record sia di classe 1.

Ora bisogna capire, per ogni previsione, se effettivamente questa abbia preceduto un guasto. Si è scelta una finestra di 90 giorni per cui considerare valida una predizione.

Si parte aggiungendo una nuova colonna, "*fail_sum*", che contiene la somma dei segnali positivi ricevuti nei 90 giorni precedenti ogni record. Questa somma è calcolata in relazione al macchinario, per evitare che i segnali riguardanti una macchina vengano interpretati anche per tutte le altre.

Questa colonna è utile per crearne un'altra, "*failure_signal*", che vale 0 se "*fail_sum*" < 1, quindi se un segnale di rottura si è verificato negli ultimi 90 giorni. Dopodiché si può dare un ID ad ogni segnale su un'ulteriore colonna, "*signal_id*", facendo la somma cumulativa di "*failure_signal*".

A questo punto, è necessario spostare i record con un segnale in un diverso dataset. La data di questi record sarà la data del segnale e l'ID sarà il macchinario a cui il segnale è rivolto. Facendo una *full outer join* con il dataset di partenza, si ottiene un dataset in cui ad ogni segnale è associata la data in cui questo è stato emesso.

Si inserisce un ultimo campo, "*warning*", che indica il numero di giorni trascorsi dal segnale al guasto, per poterlo confrontare con la finestra di predizione stabilita (90 giorni).

Per concludere, si possono definire (e contare) True Positive, False Negative e False Positive, confrontando il valore di "*warning*" con la finestra di predizione.

$$\text{Costo totale} = \text{FN} * 30.000\$ + \text{FP} * 1.500\$ + \text{TP} * 7.500\$ \qquad (12)$$

Il risultato di quest'operazione è il costo totale del modello in questione.


SMOTE
Completed
Two-Class Decision Forest
Completed
Two-Class Support Vector Machine
Completed
Tune Model Hyperparameters
Completed
Tune Model Hyperparameters
Completed
Score Model
Completed
Score Model
Completed
Select Columns in Dataset
filter columns for evaluation
Completed
Select Columns in Dataset
filter columns for evaluation
Completed
Join Data
merge with machine list
Completed
Join Data
merge with machine list
Completed
Execute Python Script
sum failures in forecast window
Completed
Execute Python Script
sum failures in forecast window
Completed
Split Data
filter all failure signals
Completed
Split Data
filter all failure signals
Completed
Select Columns in Dataset
Completed
Select Columns in Dataset
Completed
Edit Metadata
rename columns before full outer join
Completed
Edit Metadata
rename columns before full outer join
Completed
Join Data
merge signal list into scored df
Completed
Join Data
merge signal list into scored df
Completed
Enter Data Manually
Completed
Execute Python Script
find total cost
Completed
Add Columns
add model labels
Completed
Add Rows
Completed
Execute Python Script
find best model
Completed

Risultati

Poiché l'obiettivo è prettamente economico, ha senso confrontare i modelli in base al risparmio medio che porterebbero all'azienda.

Nella tabella a seguire si riportano i costi totali relativi a ciascun modello e il risparmio che si ottiene rispetto al caso peggiore.

Model	Cost	Saved
Logistic Regression	4.998.000$	2.382.000$
Support Vector Machine	5.454.000$	1.926.000$
Averaged Perceptron	5.842.500$	1.537.500$
Decision Forest	7.128.000$	252.000$
Boosted Decision Tree	7.146.000$	234.000$
Neural Network	7.309.500$	70.500$
Dummy Classifier	7.380.000$	0

Il modello Logistic Regression è quello che offre un risparmio maggiore rispetto alla manutenzione reattiva; SVM e Averaged Perceptron hanno ottenuto risultati simili ma comunque inferiori.

6 Conclusioni

Grazie a sensori sempre più economici e disponibili, il mondo dell'*Industrial Internet of Things* non è mai stato tanto avanzato; ogni impianto industriale ha la possibilità di ricevere e trasmettere informazioni ad un costo prossimo allo zero. In un panorama produttivo come questo, è evidente il vantaggio che una qualsiasi realtà industriale di dimensioni medio-grandi possa trarre utilizzando approcci di manutenzione predittiva.

Dalla prima applicazione si può concludere che, nonostante l'enorme quantità di dati mancanti del dataset di partenza ed il forte sbilanciamento delle classi, è comunque possibile allenare un modello performante grazie a tecniche come SMOTE, MICE e RUS.

La seconda applicazione dimostra meglio la convenienza a livello economico della manutenzione predittiva. Basti pensare che l'azienda avrebbe potuto risparmiare oltre 2.300.000$ in un anno predicendo i guasti tramite un modello Logistic Regression opportunamente allenato, a fronte degli oltre 7.000.000$ all'anno spesi con la strategia precedente.

La piattaforma *Azure Machine Learning* si è rivelata comoda ed intuitiva; la sua interfaccia *drag&drop* permette anche a chi non ha basi di programmazione di sperimentare con diversi modelli e dataset preimpostati, mentre chi è già esperto può comunque lavorare a livello avanzato importando script e modelli personalizzati. Un'altra comodità è l'esecuzione in cloud delle pipeline, che consente di spostare il *workload* ed eliminare il requisito di un PC potente per poter allenare i modelli.

7 Ringraziamenti

Ringrazio in primis la mia famiglia, in particolare i miei genitori, per il supporto costante che ho ricevuto per tutta la durata di questo lungo percorso: mia madre, che per ogni esame è stata in pensiero più di me; mio padre, che fino alla fine ha creduto nelle mie capacità.

Infiniti ringraziamenti vanno al mio amico e collega Bartolomeo Caruso, per la quantità industriale di spiegazioni, appunti e supporto morale, senza i quali non sarei mai riuscito ad arrivare fin qui.

Un'altra persona che ci tengo a ringraziare è Andrea Battaglia, con cui ho condiviso centinaia di ore di studio, così come tutti gli esami più memorabili della mia carriera universitaria. Lo stesso vale per Dario Camonita, terzo membro del trittico dei laureandi, nonchè compagno nella sperimentazione di software libero. Tocca poi ringraziare anche Francesco Convertino, per avermi fatto da mentore nella complicata arte degli arbitraggi.

Una menzione speciale va alla mia ragazza, Giulia Fortini, per avermi dato la motivazione finale a laurearmi dopo numerosi mesi di stallo.

Infine, è impossibile nominare uno alla volta tutti gli amici preziosi che hanno contribuito a rendere speciali questi anni, vorrei dedicare un enorme ringraziamento a ciascuno di loro.

Il mio ultimo ringraziamento va all'infinita pazienza del Prof. Cavalieri, per avermi guidato lungo il percorso di realizzazione delle applicazioni e stesura della tesi.