UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA

Analisi dati produttivi tramite tecniche di Machine Learning

Production data analysis through Machine Learning techniques

Relatore: Chiar.mo

Prof. Ciarapica Filippo Emanuele

Tesi di laurea di:

Elisei Valerio

A.A. 2019/2020

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA

Analisi dati produttivi tramite tecniche di Machine Learning

Production data analysis through Machine Learning techniques

Relatore: Chiar.mo

Prof. Ciarapica Filippo Emanuele

Tesi di laurea di:

Elisei Valerio

A.A. 2019/2020

Indice

Introduzione

Negli ultimi anni le capacità computazionali dei computer sono aumentate esponenzialmente, permettendo l'analisi di dataset di dimensioni via via sempre più grandi in minor tempo. I macchinari industriali si sono inoltre dotati di un maggior numero di sensori di controllo, anche per il graduale passaggio a Industry 4.0.

Questi fattori hanno contribuito alla crescita delle attività di ricerca e allo sviluppo di predizioni sulle linee di produzione.

Per analizzare questa grande quantità di dati raccolti ed ottenere delle previsioni affidabili si sono diffuse negli ultimi anni diverse tecniche di machine learning, tra cui gli algoritmi di decision tree e random forest, che verranno trattati nello specifico in questa tesi, applicandoli ad una linea di produzione di pompe della Magneti Marelli.

Grazie a questi si potranno andare a predire il valore di OEE, il numero di pezzi totali prodotti e il numero totale di pezzi buoni partendo dai dati ottenuti dalla linea di produzione.

In questo lavoro di tesi viene inizialmente fatta una trattazione generale sul Machine Learning, per poi passare alla descrizione degli algoritmi più importanti, concentrandosi in particolare su decision tree e random forest. Seguirà una breve presentazione dell'azienda in questione, del linguaggio Python e la descrizione del codice utilizzato per ottenere il modello predittivo finale, mostrato nella conclusione.

Il Machine Learning

Il machine learning (noto anche come apprendimento automatico) è una branca dell'intelligenza artificiale che raccoglie un insieme di metodi, sviluppati a partire dagli ultimi decenni del XX secolo in varie comunità scientifiche. Utilizza metodi statistici per migliorare progressivamente la performance di un algoritmo nell'identificare schemi nei dati.

Nell'ambito dell'informatica, l'apprendimento automatico è una variante alla programmazione tradizionale, nella quale si predispone in una macchina l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza ricevere istruzioni esplicite a riguardo.

Arthur Samuel, che coniò il termine nel 1959, identifica in linea di principio due approcci distinti. Il primo metodo, indicato come rete neurale, porta allo sviluppo di macchine ad apprendimento automatico per impiego generale, in cui il comportamento è appreso da una rete di commutazione connessa casualmente, a seguito di una routine di apprendimento basata su ricompensa e punizione (apprendimento per rinforzo).

Il secondo metodo, più specifico, consiste nel riprodurre l'equivalente di una rete altamente organizzata progettata per imparare solo alcune attività specifiche. Questa procedura, che necessita di supervisione, richiede la riprogrammazione per ogni nuova applicazione, ma risulta essere molto più efficiente dal punto di vista computazionale.

Figura 1, Arthur Samuel

L'apprendimento automatico è strettamente legato al riconoscimento di modelli e alla teoria computazionale dell'apprendimento, inoltre esplora lo studio e la costruzione di algoritmi che possano apprendere da un insieme di dati e fare delle predizioni su questi, costruendo in modo induttivo un modello basato su dei campioni. Viene poi impiegato in quei campi

dell'informatica nei quali progettare e programmare algoritmi espliciti è impraticabile; tra le possibili applicazioni si possono nominare il filtraggio delle e-mail per evitare spam, l'individuazione di intrusioni in una rete o di intrusi che cercano di violare dati, il riconoscimento ottico dei caratteri, i motori di ricerca e la visione artificiale.

L'apprendimento automatico è strettamente collegato, e spesso si sovrappone alla statistica computazionale, che si occupa dell'elaborazione di predizioni tramite l'uso di computer. È anche fortemente legato all'ottimizzazione matematica, che fornisce metodi, teorie e domini di applicazione a questo campo. Per usi commerciali, l'apprendimento automatico è conosciuto anche con il nome di analisi predittiva.

Cenni storici

L'apprendimento automatico si sviluppa con lo studio dell'intelligenza artificiale (AI), e vi è strettamente collegato: già dai primi tentativi di definire l'intelligenza artificiale come disciplina accademica, alcuni ricercatori si erano mostrati interessati alla possibilità che le macchine imparassero dai dati.

Questi ricercatori, in particolare Marvin Minsky, Arthur Samuel e Frank Rosenblatt, provarono ad avvicinarsi al problema sia attraverso vari metodi formali, sia con quelle che verranno poi definite reti neurali verso la fine degli anni '50. Si provarono a sfruttare anche ragionamenti probabilistici, in particolare nelle diagnosi mediche automatiche.

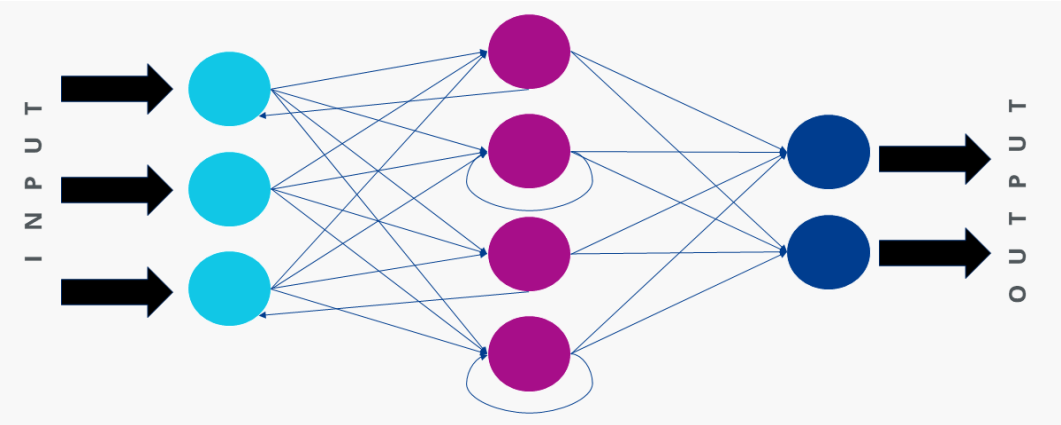


Figura 2 schema di rete neurale

Sempre negli anni '50, Alan Turing propose l'idea di una macchina che apprende, ovvero in grado di imparare e dunque diventare intelligente. La proposta specifica di Turing anticipò gli algoritmi genetici. Tuttavia, già dalla metà degli anni '50 lo studio dell'intelligenza artificiale si stava concentrando su approcci logici diversi, causando un distacco tra lo studio dell'AI e dell'apprendimento automatico. Negli anni Ottanta, i sistemi esperti dominavano il campo dell'AI, e i sistemi basati sulla statistica non venivano più studiati.

Lo studio dell'apprendimento simbolico e knowledge-based continuò nell'ambito dell'AI, portando a sviluppare la programmazione logica induttiva, ma la ricerca più prettamente statistica si svolgeva al di fuori del campo vero e proprio dell'intelligenza artificiale. Un altro motivo per cui lo studio dell'apprendimento automatico fu abbandonato fu la pubblicazione del libro "Perceptrons: an introduction to computational geometry" di Marvin Minsky e Seymour Papert, nel quale venivano descritte alcune delle limitazioni delle reti neurali e per questo subirono un significativo rallentamento.

A metà degli anni ‘80 vennero poi rivalutate, con la scoperta della backpropagation (la retropropagazione dell’errore è un algoritmo per l’allenamento delle reti neurali artificiali) e della self-organization. L’apprendimento automatico, sviluppatosi come campo di studi separato dall’AI classica, cominciò a rifiorire negli anni ‘90. Il suo obiettivo cambiò dall’ottenere l’intelligenza artificiale ad affrontare problemi risolvibili di natura pratica. Distolse inoltre la propria attenzione dagli approcci simbolici che aveva ereditato dall’AI, e si diresse verso metodi e modelli presi in prestito dalla statistica e dalla teoria della probabilità. L’apprendimento automatico ha inoltre beneficiato dalla nascita di Internet, che ha reso l’informazione digitale più facilmente reperibile e distribuibile.

Generalità

Tom M. Mitchell ha fornito la definizione più citata di apprendimento automatico nel suo libro “Machine Learning”: “Si dice che un programma apprende dall’esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l’esperienza E.” In poche parole, si potrebbe semplificare dicendo che un programma apprende se c’è un miglioramento delle prestazioni dopo un compito svolto. Questa definizione di Mitchell è rilevante, poiché fornisce una definizione operativa dell’apprendimento automatico. Fornendo questa definizione, Mitchell di fatto segue la proposta che Alan Turing fece nel suo articolo "Computing Machinery and Intelligence", sostituendo la domanda "Le macchine possono pensare?" con la domanda "Le macchine possono fare quello che noi (in quanto entità pensanti) possiamo fare?".

Figura 3 Tom M. Mitchell

Teoria dell'apprendimento

L'obiettivo principe dell'apprendimento automatico è che una macchina sia in grado di generalizzare dalla propria esperienza, ossia che sia in grado di svolgere ragionamenti induttivi.

In questo contesto, per generalizzazione si intende l'abilità di una macchina di portare a termine in maniera accurata compiti nuovi, che non ha mai affrontato, dopo aver fatto esperienza su un insieme di dati di apprendimento.

Gli esempi di addestramento si assume provengano da una qualche distribuzione di probabilità, generalmente sconosciuta e considerata rappresentativa dello spazio delle occorrenze del fenomeno da apprendere; la macchina ha il compito di costruire un modello probabilistico generale dello spazio delle occorrenze, in maniera tale da essere in grado di produrre previsioni sufficientemente accurate quando sottoposta a nuovi casi.

Gli algoritmi di machine learning vengono tipicamente classificati in quattro ampie categorie, anche dette paradigmi, secondo il tipo e la quantità di supervisione durante la fase di training sono:

- apprendimento supervisionato,
- apprendimento non supervisionato,
- apprendimento per rinforzo,
- apprendimento semi-supervisionato.

Ognuno di questi verrà successivamente approfondito in un capitolo dedicato.

Data mining e apprendimento automatico

L'apprendimento automatico viene a volte unito al data mining, che si focalizza maggiormente sull'analisi esplorativa dei dati ed utilizza principalmente il paradigma di apprendimento chiamato "apprendimento non supervisionato". Invece, l'apprendimento automatico può essere anche supervisionato.

L'apprendimento automatico e il data mining, infatti, si sovrappongono in modo significativo, ma mentre il primo si concentra sulla previsione basata su proprietà note apprese dai dati, il secondo si concentra sulla scoperta di proprietà prima sconosciute nei dati.

Il data mining sfrutta i metodi dell'apprendimento automatico, ma con obiettivi differenti; d'altro canto, l'apprendimento automatico utilizza i metodi di data mining come metodi di apprendimento non supervisionato o come passi di preprocessing per aumentare l'accuratezza dell'apprendimento. Gran parte della confusione tra le due comunità di ricerca scaturisce dall'assunzione di base del loro operato: nell'apprendimento automatico, le prestazioni sono generalmente valutate in base all'abilità di riprodurre conoscenza già acquisita, mentre in data mining il compito chiave è la scoperta di conoscenza che prima non si aveva.

Data mining

Il data mining (letteralmente dall'inglese estrazione di dati) è l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di informazioni utili da grandi quantità di dati, attraverso metodi automatici o semi-automatici e l'utilizzo scientifico, aziendale/industriale o operativo delle stesse.

Figura 4 Data mining

La statistica può anche essere definita come "estrazione di informazione utile da insiemi di dati". Il concetto di data mining è simile, ma con una sostanziale differenza: la statistica permette di elaborare informazioni generali riguardo ad una popolazione (es. percentuali di disoccupazione, nascite), mentre il data mining viene utilizzato per cercare correlazioni tra più variabili relativamente ai singoli individui; ad esempio conoscendo il comportamento medio dei clienti di una compagnia telefonica cerco di prevedere quanto spenderà il cliente medio nell'immediato futuro.

In sostanza il data mining è l'analisi, da un punto di vista matematico, eseguita su banche dati di grandi dimensioni, preceduta tipicamente da altre fasi di preparazione, trasformazione, filtraggio dei dati come il data cleaning.

Oggi il data mining ha una duplice funzione:

- estrazione, con tecniche analitiche di informazioni implicite all'avanguardia, nascoste, da dati già strutturati, per renderle disponibili e direttamente utilizzabili;
- esplorazione ed analisi, eseguita in modo automatico o semiautomatico, su grandi quantità di dati al fine di scoprire pattern (schemi o regolarità) significativi.

In entrambi i casi i concetti di informazione e di significato sono legati strettamente al dominio applicativo in cui si esegue il data mining. In altre parole un dato può essere interessante o trascurabile a seconda del tipo di applicazione in cui si vuole operare.

Questo tipo di attività è cruciale in molti ambiti della ricerca scientifica, ma anche in altri settori (per esempio in quello delle ricerche di mercato). Nel mondo professionale è utilizzato per risolvere problematiche diverse tra loro, che vanno dalla gestione delle relazioni con i clienti, all'individuazione di comportamenti fraudolenti, fino all'ottimizzazione di siti web.

Ottimizzazione e apprendimento automatico

L'apprendimento automatico ha legami molto stretti con l'ottimizzazione: molti problemi di apprendimento sono formulati come la ricerca del minimo di una qualche funzione di costo su un insieme di esempi di apprendimento. La funzione di costo (o funzione di perdita) rappresenta la differenza tra le previsioni del modello che si sta "addestrando" e il problema reale. Le differenze tra i due campi (l'apprendimento automatico e l'ottimizzazione) sorgono dall'obiettivo della generalizzazione: mentre gli algoritmi di ottimizzazione possono minimizzare la perdita su un insieme di apprendimento, l'apprendimento automatico si preoccupa di minimizzare la perdita su campioni mai visti dalla macchina.

Soft computing e apprendimento automatico

La risoluzione automatica di problemi avviene, nel campo dell'informatica, in due modi differenti: tramite paradigmi di hard computing o tramite paradigmi di soft computing. Per hard computing si intende la risoluzione di un problema tramite l'esecuzione di un algoritmo ben definito e decidibile. La maggior parte dei paradigmi di hard computing sono metodi ormai consolidati, ma presentano alcuni lati negativi: richiedono sempre un modello analitico preciso e definibile, e spesso un alto tempo di computazione.

Le tecniche di soft computing si prefiggono invece di valutare, decidere, controllare e calcolare in un ambito impreciso e vago emulando e utilizzando la capacità degli esseri umani di eseguire attività sulla base della loro esperienza. Il soft computing si avvale delle caratteristiche delle sue tre principali branche:

- la possibilità di modellare e di controllare sistemi incerti e complessi, nonché di rappresentare la conoscenza in maniera efficiente attraverso le descrizioni linguistiche tipiche della teoria degli insiemi sfocati (insiemi fuzzy);
- la capacità d'ottimizzazione degli algoritmi genetici la cui computazione si ispira alle leggi di selezione e mutazione tipiche degli organismi viventi;
- la capacità di apprendere complesse relazioni funzionali delle reti neurali, ispirate a quelle proprie dei tessuti cerebrali.

In generale, la logica fuzzy, le reti neurali e gli algoritmi genetici possono considerarsi i principali costituenti di ciò che potrebbe essere definito soft computing. A differenza dei metodi di calcolo tradizionali o hard, il soft computing si prefigge lo scopo di adattarsi all'imprecisione del mondo reale. Il suo principio guida può così esprimersi: sfruttare la tolleranza per l'imprecisione, l'incertezza e le verità parziali in modo da ottenere trattabilità, robustezza e soluzioni a basso costo. Nei prossimi anni, il soft computing è probabilmente destinato a giocare un ruolo sempre più rilevante nella concezione e progettazione di sistemi il cui MIQ (Quoziente Intellettivo di Macchina) sia di gran lunga più alto di quello dei sistemi convenzionali.

L'apprendimento automatico si avvale delle tecniche di soft computing.

Esempi di applicazioni pratiche

Riconoscimento vocale del testo

Tutti i sistemi di riconoscimento vocale di maggior successo utilizzano metodi di apprendimento automatico. Ad esempio, il SPHINXsystem impara le strategie di altoparlanti specifici per riconoscere i suoni primitivi (fonemi) e le parole del segnale vocale osservato. Metodi di apprendimento basati su reti neurali e su modelli di Markov nascosti sono efficaci per la personalizzazione automatica di vocabolari, caratteristiche del microfono, rumore di fondo, ecc.

Guida automatica di veicoli

Metodi di apprendimento automatico sono stati usati per addestrare i veicoli controllati da computer. Ad esempio, il sistema ALVINN ha usato le sue strategie per imparare a guidare senza assistenza a 70 miglia all'ora per 90 miglia su strade pubbliche, tra le altre auto. Con tecniche simili sono possibili applicazioni in molti problemi di controllo basato su sensori.

Classificazione di nuove strutture astronomiche

Metodi di apprendimento automatico sono stati applicati ad una varietà di banche dati di grandi dimensioni per imparare regolarità generali implicito nei dati. Ad esempio, algoritmi di apprendimento basati su alberi di decisione sono stati usati dalla NASA per classificare oggetti celesti a partire dal secondo Palomar Observatory Sky Survey. Questo sistema è oggi utilizzato per classificare automaticamente tutti gli oggetti nel Sky Survey, che si compone di tre terabyte di dati immagine.

Etica

L'apprendimento automatico solleva un numero di problematiche etiche. I sistemi addestrati con insiemi di dati faziosi o pregiudizievoli possono esibire questi pregiudizi quando vengono interpellati: in questo modo possono essere digitalizzati pregiudizi culturali quali il razzismo istituzionale e il classismo. Di conseguenza la raccolta responsabile dei dati può diventare un aspetto critico dell'apprendimento automatico.
In ragione dell'innata ambiguità dei linguaggi naturali, le macchine addestrate su corpi linguistici necessariamente apprenderanno questa ambiguità.

Apprendimento supervisionato

L'apprendimento supervisionato (Supervised Learning) è una tecnica di apprendimento automatico che mira a istruire un sistema informatico in modo da consentirgli di elaborare automaticamente una previsione sui valori di uscita di un sistema rispetto ad un input, sulla base di una serie di esempi ideali, costituiti da coppie di input e di output, che gli vengono inizialmente forniti. Il sistema impara il nesso tra loro e ne estrapola una regola riutilizzabile per altri compiti simili.

Algoritmi di apprendimento supervisionato possono essere utilizzati nei più disparati settori. Degli esempi riguardano il campo medico in cui si può prevedere lo scatenarsi di particolari crisi sulla base dell'esperienza di passati dati biometrici, l'identificazione vocale che migliora sulla base degli ascolti audio passati, l'identificazione della scrittura manuale che si perfeziona sulle osservazioni degli esempi sottoposti dall'utente.

La casistica degli output può essere molto varia; ciononostante differenzia l'apprendimento di valori quantitativi (comunemente chiamata "regressione") da valori qualitativi (chiamata "classificazione").

Classificazione: la macchina viene addestrata alla classificazione dal supervisore tramite l'aggiunta di etichette sui dati in cui giudica il risultato. Ogni etichetta è una classe discreta che identifica il risultato atteso (es. spam o no spam) oppure un giudizio di valore.

Regressione lineare: il risultato (output) è un valore continuo. Alla macchina spetta il compito di trovare una relazione tra i valori di input (valori descrittivi) e di output.

Nell'apprendimento supervisionato la macchina deve stimare una funzione f(x) incognita che collega le variabili di input x a una variabile di output y.

$$y = f(x)$$

La macchina non conosce la funzione f(x), il suo scopo è quindi quello di stimare una funzione ipotesi h(x) in grado di approssimare la f(x).

$$y = h(x)$$

Per farlo analizza un insieme di dati detto training set fornito dal supervisore, ossia un insieme di addestramento composto da un insieme di coppie (x, y).

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	y
1	0	0	1	0	0	1	1	0	0	1
1	0	1	1	0	0	1	1	0	1	0
0	1	1	0	1	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1
1	0	1	1	1	0	1	0	0	1	0
0	0	1	1	0	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	0	0
1	1	1	0	1	1	1	0	1	0	1

Figura 5 esempio di dataset

Pertanto, l'input di un algoritmo di machine learning supervisionato è una matrice con esempi etichettati, a partire da questi dati la macchina elabora la funzione ipotesi h(x).
Per capire se la funzione ipotesi è corretta la macchina deve valutarne l'accuratezza, quindi capire se essa approssima o meno la funzione f(x) senza di fatto conoscerla.
Per capirlo utilizza un altro insieme di dati, detto test set, fornito sempre dal supervisore.

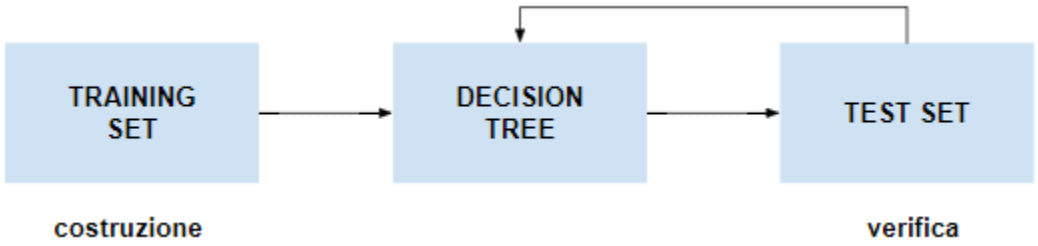


Figura 6 schema di funzionamento

A questo punto la macchina risponde agli N_t esempi del test set, per poi confrontare ogni sua risposta R con la risposta corretta indicata (Y) nel test set.

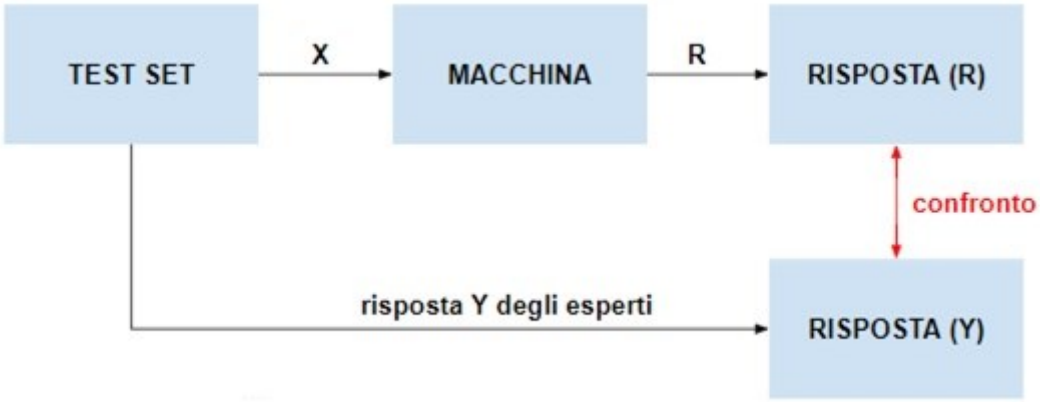


Figura 7 schema di risposta

Le risposte coincidenti (R=Y) incrementano il numero delle risposte corrette R_c della macchina.

Se la percentuale di risposte corrette R_c/N_t della macchina è soddisfacente, la funzione ipotesi h(x) supera l'esame e viene accolta.

$$\frac{R_C}{N_T} = \frac{risposte\ \ corrette}{numero\ \ test}$$

In caso contrario l'apprendimento supervisionato riparte con l'analisi di un ulteriore training set ed il ciclo ricomincia da capo.

Molti di questi algoritmi funzionano in maniera efficiente se lavorano in un mondo lineare, presupponendo che ad ingressi simili corrispondano uscite simili. Esistono molte condizioni in cui una simile approssimazione è accettabile, ma non sempre è così. La stima della funzione ipotesi serve ad attenuare le problematiche che derivano dalla trattazione di problemi non completamente lineari.

Si può facilmente intuire che il funzionamento corretto ed efficiente di questi algoritmi dipende in modo significativo dall'esperienza; se si fornisce poca esperienza, l'algoritmo potrebbe non creare una funzione interna efficiente, mentre con esperienza eccessiva la funzione interna potrebbe divenire molto complessa tanto da rendere lenta l'esecuzione.

Questi algoritmi sono molto sensibili al rumore, anche pochi dati errati potrebbero rendere l'intero sistema non affidabile e condurlo a decisioni errate. Una soluzione a questo problema è quello di associarli a controllori che si basano sulla logica fuzzy.

Tradizionalmente i principali algoritmi sono:

- albero di decisione;
- regole di decisione;
- sistemi esperti.

La ricerca oggi si concentra su quelle che sono considerate le due classi principali di algoritmi possibili: metodi generativi e metodi discriminativi.

I metodi generativi si basano sulla creazione di un modello dei dati che poi viene utilizzato per predire le risposte desiderate (o dati di uscita). Esempi sono le reti bayesiane o più in generale i modelli grafici.

I metodi discriminativi, al contrario, cercano di modellare direttamente la relazione tra dati in entrata e quelli in uscita, in modo da minimizzare una funzione obiettivo (loss function). Esempi di questo tipo di modello sono le macchine a vettori di supporto (Support Vector Machines) e più in generale i metodi basati su funzioni di kernel.

Apprendimento non supervisionato

L'apprendimento non supervisionato (Unsupervised Learning) è quando al sistema vengono forniti solo set di dati senza alcuna indicazione del risultato desiderato. Lo scopo di questo secondo metodo di apprendimento è "risalire" a schemi e modelli nascosti, ossia identificare negli input una struttura logica senza che questi siano preventivamente etichettati.

Negli algoritmi di apprendimento non supervisionato il training set non è etichettato, cioè non contiene i valori di output tramite i quali allenare il modello e, quindi, non sono note le classi dei pattern utilizzati per l'addestramento.

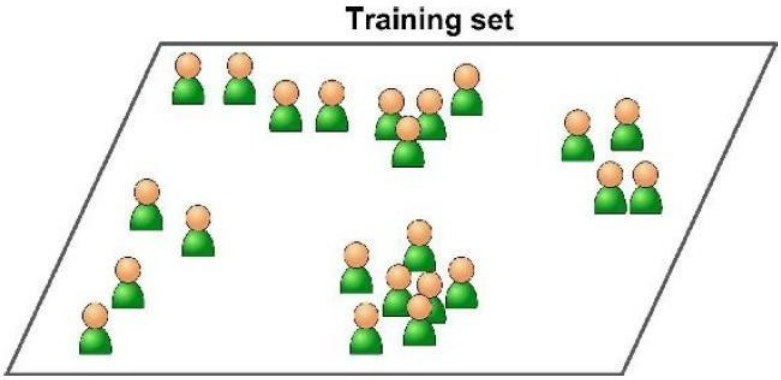


Figura 8 training set

Un esempio tipico di questi algoritmi lo si ha nei motori di ricerca. Questi programmi, data una o più parole chiave, sono in grado di creare una lista di link rimandanti alle pagine che l'algoritmo di ricerca ritiene attinenti alla ricerca effettuata. La validità di questi algoritmi è legata alla utilità delle informazioni che riescono ad estrarre dalla base di dati, nell'esempio sopracitato è legata all'attinenza dei link con l'argomento cercato.

Questi algoritmi lavorano confrontando i dati e ricercando similarità o differenze. Sono molto efficienti con elementi di tipo numerico, dato che possono utilizzare tutte le tecniche derivate dalla statistica, ma sono molto meno efficienti con dati non numerici. Se i dati sono dotati di un ordinamento intrinseco gli algoritmi riescono comunque ad estrarre informazioni, ma se i dati in ingresso non sono dotati di un qualche tipo di ordinamento spesso gli algoritmi falliscono. Se i dati non sono dotati di ordinamento cercare di ordinarli imponendo una graduatoria arbitraria non risolve il problema. Questo si può facilmente capire con un esempio. Supponiamo di disporre di un database con l'elenco dei colori utilizzati da uno stilista: si potrebbe cercare di associare ad ogni colore uno specifico numero e su quello fare delle analisi di tipo statistico. Ma dato che l'associazione tra colore

e numero è arbitraria si possono pensare ad infinite associazioni che darebbero infiniti risultati diversi. Questi algoritmi in conclusione lavorano correttamente in presenza di dati contenenti un ordinamento o un raggruppamento netto e chiaramente identificabile.
Tra i più importanti algoritmi di questa tipologia possiamo trovare:

- clustering: individua gruppi (cluster) di pattern con caratteristiche simili in cui le classi del problema non sono note e i pattern non etichettati (senza output). Spesso il numero di cluster non è noto a priori, ma quelli individuati nell'apprendimento possono poi essere usati come classi. La natura non supervisionata del problema li rende più complicati di quelli di classificazione.

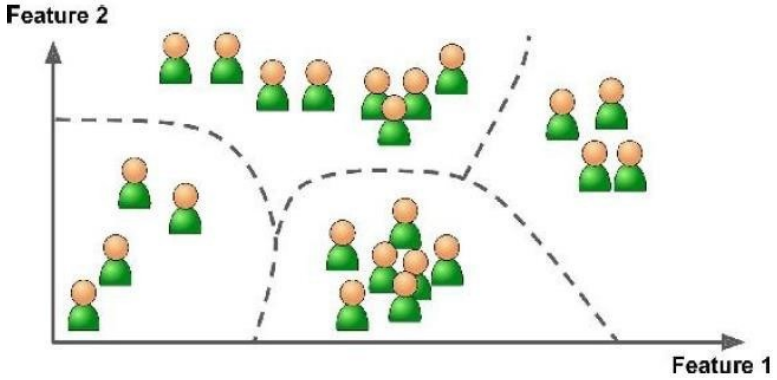


Figura 9 esempio di clustering

- riduzione di dimensionalità: riduce il numero di dimensioni dei pattern in input, senza perdere troppe informazioni. L'operazione comporta una perdita di informazione, ma l'obiettivo è mantenere quelle importanti per il caso, dipendenti quindi dall'applicazione. Risulta quindi molto utile per rendere trattabili problemi con dimensionalità molto elevata, scartando informazioni ridondanti e/o instabili, permettendo quindi al codice di calcolare più velocemente grazie al minor spazio su disco occupato dai dati e, talvolta, avere prestazioni più alte.

Apprendimento per rinforzo

L'apprendimento per rinforzo (reinforcement learning) è uno dei tre paradigmi principali dell'apprendimento automatico, insieme all'apprendimento supervisionato e a quello non supervisionato.

Un algoritmo di reinforcement learning ha come obiettivo l'apprendimento di un comportamento ottimale a partire dalle esperienze passate, occupandosi quindi di problemi di decisioni sequenziali.

L'acquisizione della conoscenza è dedicata ad un agente, il quale osserva l'ambiente circostante ed esegue azioni per modificarlo, provocando passaggi da uno stato all'altro e ricevendo delle ricompense, dette rewards, che possono essere anche delle penalties (ricompense negative). L'obiettivo è apprendere l'azione ottimale in ciascuno stato, in modo da massimizzare la somma delle ricompense ottenute nel lungo periodo.

La qualità di un'azione è data da un valore numerico di "ricompensa", ispirata al concetto di rinforzo, che ha lo scopo di incoraggiare comportamenti corretti dell'agente. Questo tipo di apprendimento è solitamente modellizzato tramite i processi decisionali di Markov e può essere effettuato con diverse tipologie di algoritmi, classificabili in base all'utilizzo di un modello che descriva l'ambiente, alle modalità di raccolta dell'esperienza (in prima persona o da parte di terzi), al tipo di rappresentazione degli stati del sistema e delle azioni da compiere (discreti o continui).

Questa tecnica si basa sul presupposto che all'interno di un sistema si possano predisporre:

- un meccanismo logico A in grado di scegliere degli output sulla base degli input ricevuti;
- un meccanismo logico B in grado di valutare l'efficacia degli output rispetto ad un preciso parametro di riferimento;
- un meccanismo logico C capace di cambiare il meccanismo A per massimizzare la valutazione di efficacia effettuata da B.

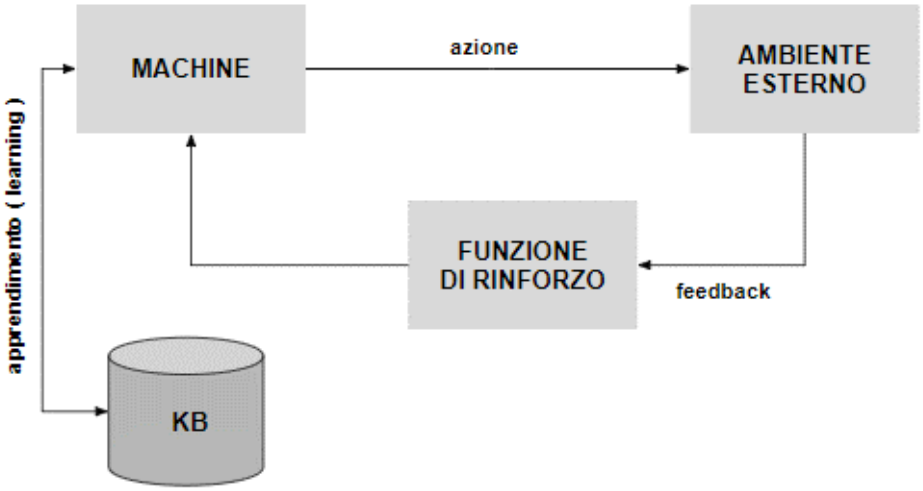


Figura 10 meccanismo di apprendimento per rinforzo

Il modo in cui questi meccanismi dovrebbero collaborare è descritto dai seguenti punti:

- se il meccanismo A effettua una scelta efficace allora il meccanismo B manda in output un premio proporzionale all'efficacia della scelta di A;
- se il meccanismo A effettua una scelta inefficace allora il meccanismo B manda in output una penalità proporzionale all'inefficacia della scelta di A;
- il meccanismo C, osservando l'agire di A e B, cerca di modificare la funzione matematica che regola il comportamento di A in modo da massimizzare la quantità e la qualità dei "premi".

I meccanismi B e C sono quelli che vanno a costituire il metodo di rinforzo proprio di questa metodica di apprendimento. Per attuare i meccanismi ed i comportamenti descritti nelle righe precedenti, dal punto di vista logico, si necessita delle seguenti componenti:

- insieme di input: rappresenta i possibili input che il sistema può ricevere (servono per determinare lo stato del sistema);
- funzione valore di stato: questa funzione associa un parametro di valutazione ad ogni stato del sistema;
- funzione valore di azione: questa funzione associa un parametro di valutazione ad ogni possibile coppia stato-azione;
- tecnica di rinforzo: consiste in una funzione di rinforzo che, a seconda delle prestazioni attuali e dell'esperienza passata, fornisce delle direttive con cui cambiare la funzione di valore di stato e la funzione di valore d'azione;
- insieme di output: rappresenta le possibili decisioni che il sistema può intraprendere.

Gli input al sistema possono provenire dai più svariati sensori. Ad esempio, nel caso di un robot che deve imparare a muoversi all'interno di un percorso, gli input potrebbero essere forniti da dei sensori di prossimità che dovrebbero essere poi rimappati in opportuni stati che nel caso dell'esempio potrebbero essere "ostacolo di fronte", "strada libera", "muro sul lato" ecc. Per mappare i valori dei sensori a particolari stati si sono rivelate particolarmente efficaci le tecniche basate su controllori fuzzy.

Apprendimento semi supervisionato

L'apprendimento semi supervisionato è un modello ibrido, che si snoda tra l'apprendimento supervisionato e quello non supervisionato. Questi modelli mirano a utilizzare una piccola quantità di dati di allenamento etichettati insieme a una grande quantità di dati di allenamento senza etichetta.

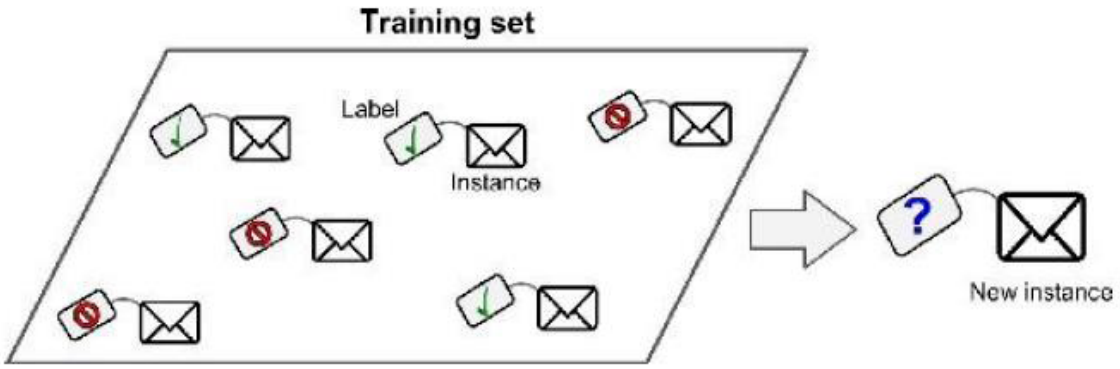


Figura 11 training set di apprendimento semi supervisionato

Ciò si verifica spesso in situazioni reali in cui l'etichettatura dei dati è molto costosa e/o si dispone di un flusso costante di dati. L'obiettivo di fondo è sempre lo stesso: identificare regole e funzioni per la risoluzione dei problemi, nonché modelli e strutture di dati utili a raggiungere determinati obiettivi.

Ad esempio, se stessimo cercando di rilevare messaggi inappropriati in un social network, non ci sarebbe modo di ottenere informazioni etichettate a mano su ogni messaggio, poiché ce ne sono semplicemente troppe e sarebbe troppo costoso. Invece, possiamo etichettare a mano un sottoinsieme di essi e sfruttare le tecniche semi-supervisionate per utilizzare questo piccolo set di dati etichettati per aiutarci a comprendere il resto del contenuto dei messaggi appena arrivano.

Alcuni metodi semi-supervisionati comuni sono le macchine vettoriali di supporto trasversali e i metodi basati su grafici, ad esempio la propagazione delle etichette.

I metodi semi-supervisionati devono fare alcune ipotesi (presupposti) sui dati al fine di giustificare l'utilizzo di una piccola serie di dati etichettati per trarre conclusioni sui punti dati non etichettati. Questi possono essere raggruppati in tre categorie.

La prima categoria riguarda il presupposto di continuità: si presume che i punti dati "vicini" tra loro abbiano maggiori probabilità di avere un'etichetta comune.

Il secondo presupposto è l'ipotesi del cluster: si presume che i dati formino naturalmente cluster discreti e che i punti nello stesso cluster abbiano maggiori probabilità di condividere un'etichetta.
La terza categoria riguarda il presupposto molteplice: si presume che i dati si trovino approssimativamente in uno spazio di dimensioni inferiori (o collettore) rispetto allo spazio di input. Questo scenario è rilevante quando un sistema non osservabile o difficile da osservare con un numero ridotto di parametri produce output osservabile ad alta dimensione.

Approcci

Di seguito sono elencati i principali metodi di machine learning, sia di apprendimento supervisionato che non supervisionato.

Programmazione logica induttiva

La programmazione logica induttiva (acronimo ILP, dall'inglese Inductive Logic Programming) è una sotto area dell'apprendimento automatico che utilizza programmazione logica come una rappresentazione uniforme per esempi, conoscenze di base e ipotesi. Data una codifica della nota conoscenza di base e un insieme di esempi rappresentati come fatti in una base di dati logica, un sistema ILP deriva un programma logico ipotetico da cui conseguono tutti gli esempi positivi, e nessuno di quelli negativi.
Schema: esempi positivi + esempi negativi + conoscenze di base ⇒ ipotesi .
Spesso viene usato come sinonimo di apprendimento relazionale, ossia apprendimento nel contesto di rappresentazioni equivalenti alla logica del primo ordine. La programmazione induttiva è un campo simile che considera ogni tipo di linguaggio di programmazione per rappresentare le ipotesi invece che soltanto la programmazione logica, come ad esempio programmi funzionali.

Regole di associazione

L'apprendimento automatico basato su regole di associazione è un metodo di apprendimento che identifica, apprende ed evolve delle "regole" con l'intento di immagazzinare, manipolare e applicare conoscenza.
La caratteristica principale di questo tipo di apprendimento è l'identificazione e l'utilizzo di un insieme di regole relazionali che rappresentano nel loro insieme la conoscenza catturata dal sistema. Ciò si pone in controtendenza con altri tipi di apprendimento automatico che normalmente identificano un singolo modello che può essere applicato universalmente ad ogni istanza per riuscire a fare su di essa una previsione.
R. Agrawal, Imielinski, A. Swami introdussero le regole di associazione per la scoperta di regolarità all'interno delle transazioni registrate nelle vendite dei supermercati. Per esempio, la regola individuata nell'analisi degli scontrini di un supermercato indica che il se il cliente compra insieme cipolle e patate è probabile che acquisti anche della carne per hamburger. Tale informazione può essere utilizzata come base per le decisioni riguardanti le attività di marketing, come ad esempio le offerte promozionali o il posizionamento dei

prodotti negli scaffali. Le regole di associazione sono anche usate in molte altre aree, quali il Web mining, la scoperta di anomalie e la bioinformatica.

Reti neurali artificiali

Le reti neurali artificiali sono modelli matematici composti da neuroni artificiali di ispirazione alle reti neurali biologiche (quella umana o animale) e vengono utilizzate per risolvere problemi ingegneristici di Intelligenza Artificiale legati a diversi ambiti tecnologici come l'informatica, l'elettronica, la simulazione o altre discipline.

Volendo dare una definizione più dettagliata si può affermare che le reti neurali sono modelli di calcolo matematico-informatici basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni; tali interconnessioni derivano da neuroni artificiali e processi di calcolo basati sul modello delle scienze cognitive chiamato "connessionismo". Vale a dire basati su PDP – Parallel Distributed Processing, elaborazione a parallelismo distribuito delle informazioni: il cervello umano elabora le informazioni dei vari sensi in modo parallelo e distribuisce le informazioni in tutti i vari nodi della rete, non in una memoria centrale. Facendo il paragone con l'informatica tradizionale, i calcoli avvengono in modo seriale e non in parallelo e i dati vengono immagazzinati in una memoria centrale.

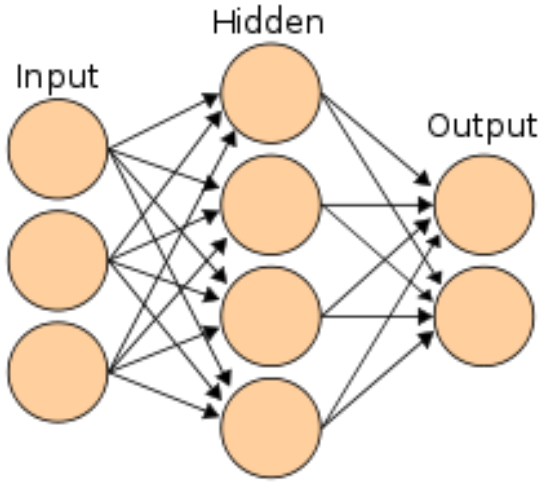


Figura 12 schema di rete neurale

Nell'analisi del cosa sono e come funzionano le reti neurali non si possono trascurare i sistemi esperti. Si tratta per lo più di applicazioni che rientrano nella branca dell'intelligenza artificiale perché riproducono artificialmente le prestazioni di una persona esperta di un determinato dominio di conoscenza o campo di attività.

Ciò che differenzia i sistemi esperti dai normali programmi software sono i dati che costituiscono la knowledge base: anziché "appoggiarsi" ad una struttura decisionale predefinita, i sistemi esperti riescono a proporre all'utente la migliore delle alternative possibili trovando la soluzione ottimale al problema tra tutte quelle disponibili.

Una rete neurale invece si presenta come un sistema "adattivo" in grado di modificare la sua struttura (i nodi e le interconnessioni) basandosi sia su dati esterni sia su informazioni interne che si connettono e passano attraverso la rete neurale durante la fase di apprendimento e ragionamento.

Capiamo come: una rete neurale biologica riceve dati e segnali esterni; questi vengono elaborati in informazioni attraverso un imponente numero di neuroni (che rappresentano la capacità di calcolo) interconnessi tra loro in una struttura non-lineare e variabile in risposta a quei dati e stimoli esterni stessi.

Allo stesso modo, le reti neurali artificiali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione: ricevono segnali esterni su uno strato di nodi (che rappresenta l'unità di elaborazione, il processore); ognuno di questi "nodi d'ingresso" è collegato a svariati nodi interni della rete che, tipicamente, sono organizzati a più livelli in modo che ogni singolo nodo possa elaborare i segnali ricevuti trasmettendo ai livelli successivi il risultato delle sue elaborazioni (quindi delle informazioni più evolute, dettagliate).

In linea di massima, le reti neurali sono formate da tre strati (che però possono coinvolgere migliaia di neuroni e decine di migliaia di connessioni):

- lo strato degli ingressi (I – Input): è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete;
- il cosiddetto strato H – hidden (strato nascosto): è quello che ha in carica il processo di elaborazione vero e proprio (e può anche essere strutturato con più colonne-livelli di neuroni);
- lo strato di uscita (O – Output): qui vengono raccolti i risultati dell'elaborazione dello strato H e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

Affinché questo processo risulti performante è necessario "addestrare" le reti neurali, ossia fare in modo che apprendano come comportarsi nel momento in cui andrà risolto un problema ingegneristico, come per esempio il riconoscimento di un essere umano dall'analisi delle immagini (attraverso, per esempio, la tecnologia del riconoscimento facciale).

I sistemi esperti vengono spesso associati alle reti neurali artificiali, mentre in realtà la differenza tra queste tecnologie è abbastanza evidente quando si mettono a confronto le funzionalità:

- un sistema esperto può dedurre alcuni ragionamenti (Mario è un essere umano; gli uomini sono mortali; Mario non è immortale), cosa che la rete neurale non fa; quest'ultima però è perfettamente in grado di riconoscere un volto umano in una immagine complessa, cosa che il sistema esperto non è in grado di fare;
- il sistema esperto è in grado di spiegare in che modo è arrivato ad una soluzione (ripercorrendo dove e come ha applicato le regole If-Then sulla base di conoscenza); il funzionamento delle reti neurali è invece molto complesso ed è quasi impossibile risalire al procedimento logico che ha portato ad una determinata soluzione;
- i sistemi esperti necessitano di un primario intervento da parte dell'esperto di dominio che definisce le regole che alimentano il motore inferenziale, devono quindi essere programmati; le reti neurali invece "deducono" regole e attività in modo automatico, attraverso l'apprendimento (le reti neurali non usano regole If-Then);

Nonostante oggi l'attenzione sia spesso rivolta solo alle reti neurali, la maturità tecnologica raggiunta dai sistemi esperti ha consentito loro di trovare importanti sbocchi applicativi, tanto che oggi si parla del "next level", quello che vede sistemi esperti e reti neurali lavorare insieme.

Programmazione genetica

La programmazione genetica (GP) è una metodologia di programmazione automatizzata, ispirata dall'evoluzione biologica, per scoprire programmi informatici che svolgano in maniera ottimale un determinato compito.

È una particolare tecnica di apprendimento automatico che usa un algoritmo evolutivo per ottimizzare una popolazione di programmi di computer secondo un paesaggio adattativo determinato dall'abilità del programma di arrivare a un risultato computazionalmente valido (ovvero di saper svolgere il compito dato).

I primi esperimenti con la GP sono stati eseguiti da Stephen F. Smith (1980) e Nichael L. Cramer (1985). Sino agli anni '90, a causa delle grandi risorse di calcolo necessarie, il ricorso alla GP era limitato a problemi relativamente semplici. In seguito, col rapido

migliorare delle prestazioni dei processori e grazie a sviluppi nella tecnologia GP, si sono ottenuti interessanti risultati in numerose aree.
Lo sviluppo di una teoria per la programmazione genetica fu compito arduo, cosa per cui, negli anni '90, fu considerata una sorta di paria tra le varie tecniche di ricerca. Tuttavia, dopo una serie di risultati positivi nei primi anni dopo il 2000, la teoria della GP ha avuto un formidabile e rapido sviluppo; così rapido che è stato possibile costruire modelli probabilistici esatti della GP (teorie, diagrammi e modelli tramite catene di Markov) e mostrare che la GP è più generale e come tale include gli algoritmi genetici.
I programmi creati con la GP possono essere scritti in molti linguaggi di programmazione. Nelle prime e tradizionali implementazioni della GP le istruzioni e i dati erano organizzati in strutture ad albero, quindi si preferiva l'uso di linguaggi che avessero queste strutture come tipo di dato primitivo; un esempio importante di linguaggio utilizzato da Koza è il Lisp. Sono state suggerite e implementate con successo anche altre forme di GP, come la più semplice rappresentazione lineare che ben si adatta ai normali linguaggi imperativi.
Gli algoritmi genetici sono stati applicati con successo a una varietà di compiti di apprendimento e di altri problemi di ottimizzazione. Ad esempio, essi sono stati usati per imparare raccolte di norme per il controllo del robot e per ottimizzare la topologia dei parametri di apprendimento per reti neurali artificiali.

Reti bayesiane

Il ragionamento bayesiano fornisce un approccio probabilistico di inferenza. Esso si basa sul presupposto che le quantità di interesse sono disciplinate da distribuzioni di probabilità e che le decisioni ottimali possono essere prese a seguito dell'analisi di queste probabilità insieme ai dati osservati. Nell'ambito dell'apprendimento automatico, la teoria Bayesiana è importante perché fornisce un approccio quantitativo per valutare le prove a sostegno dell'ipotesi alternativa. Il Ragionamento bayesiano fornisce la base per l'apprendimento negli algoritmi che manipolano direttamente le probabilità.
Una rete Bayesiana (BN, Bayesian network) è un modello grafico probabilistico che rappresenta un insieme di variabili stocastiche - possono essere quantità osservabili, variabili latenti, parametri sconosciuti o ipotesi - con le loro dipendenze condizionali attraverso l'uso di un grafo aciclico diretto (DAG). Gli archi rappresentano condizioni di dipendenza; i nodi che non sono connessi rappresentano variabili che sono condizionalmente indipendenti tra di loro. Ad ogni nodo è associata una funzione di probabilità che prende in input un particolare insieme di valori per le variabili del nodo

genitore e restituisce la probabilità della variabile rappresentata dal nodo. Per esempio, se i genitori del nodo sono variabili booleane allora la funzione di probabilità può essere rappresentata da una tabella in cui ogni entry rappresenta una possibile combinazione di valori vero o falso che i suoi genitori possono assumere. Esistono algoritmi efficienti che effettuano inferenza e apprendimento a partire dalle reti Bayesiane.

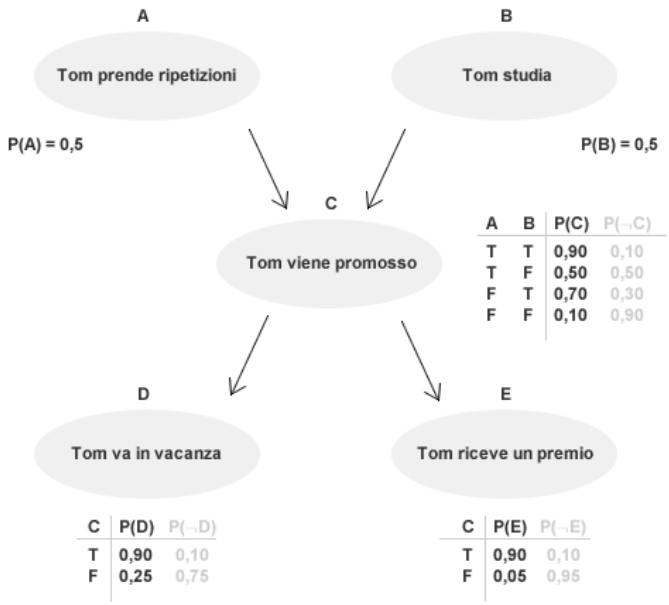


Figura 13 esempio di rete bayesiana

Per esempio, una rete Bayesiana potrebbe rappresentare la relazione probabilistica esistente tra i sintomi e le malattie. Dati i sintomi, la rete può essere usata per calcolare la probabilità della presenza di diverse malattie.

Il termine modello gerarchico è talvolta considerato un particolare tipo di rete Bayesiana, ma non ha nessuna definizione formale. Qualche volta viene usato per modelli con tre o più livelli di variabili stocastiche; in altri casi viene usato per modelli con variabili latenti. Comunque, in generale qualsiasi rete Bayesiana moderatamente complessa viene usualmente detta "gerarchica".

Macchine a vettori di supporto

Le macchine a vettori di supporto (SVM, dall'inglese Support Vector Machines) sono dei modelli di apprendimento supervisionato associati ad algoritmi di apprendimento per la regressione e la classificazione.

Dato un insieme di esempi per l'addestramento, ognuno dei quali etichettato con la classe di appartenenza fra le due possibili classi, un algoritmo di addestramento per le SVM

costruisce un modello che assegna i nuovi esempi a una delle due classi, ottenendo quindi un classificatore lineare binario non probabilistico. Un modello SVM è una rappresentazione degli esempi come punti nello spazio, mappati in modo tale che gli esempi appartenenti alle due diverse categorie siano chiaramente separati da uno spazio il più possibile ampio. I nuovi esempi sono quindi mappati nello stesso spazio e la predizione della categoria alla quale appartengono viene fatta sulla base del lato nel quale ricade.

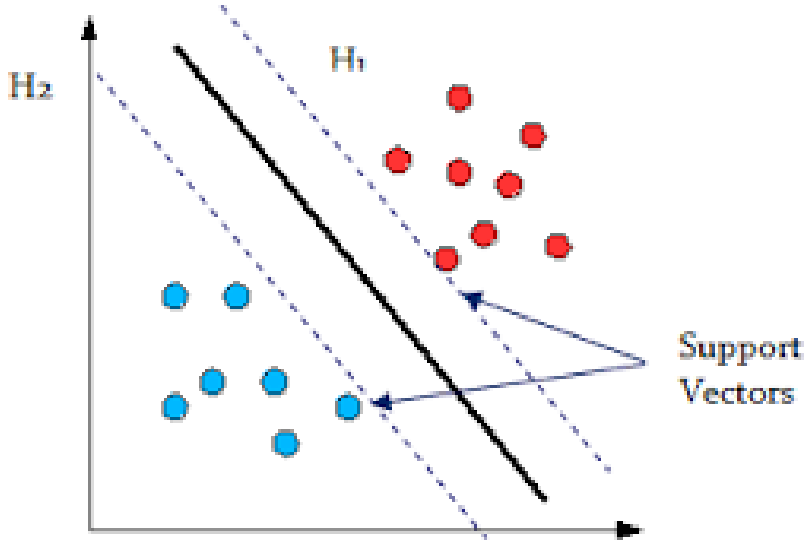


Figura 14 esempio macchina a vettori di supporto

Oltre alla classificazione lineare è possibile fare uso delle SVM per svolgere efficacemente la classificazione non lineare utilizzando il metodo kernel, mappando implicitamente i loro ingressi in uno spazio delle caratteristiche multidimensionali.

Quando gli esempi non sono etichettati è impossibile addestrare in modo supervisionato e si rende necessario l'apprendimento non supervisionato: questo approccio cerca d'identificare i naturali gruppi in cui si raggruppano i dati, mappando successivamente i nuovi dati nei gruppi ottenuti. L'algoritmo di raggruppamento a vettori di supporto, creato da Hava Siegelmann e Vladimir N. Vapnik, applica le statistiche dei vettori di supporto, sviluppate negli algoritmi delle SVM, per classificare dati non etichettati, ed è uno degli algoritmi di raggruppamento maggiormente utilizzato nelle applicazioni industriali.

Apprendimento profondo

La discesa dei prezzi per l'hardware e lo sviluppo di GPU per uso personale negli ultimi anni hanno contribuito allo sviluppo del concetto di apprendimento profondo.

L'apprendimento profondo (in inglese deep learning) è quel campo di ricerca dell'apprendimento automatico e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso. In altre parole, secondo la definizione dell'Osservatorio Artificial Intelligence del Politecnico di Milano, per apprendimento profondo si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa.

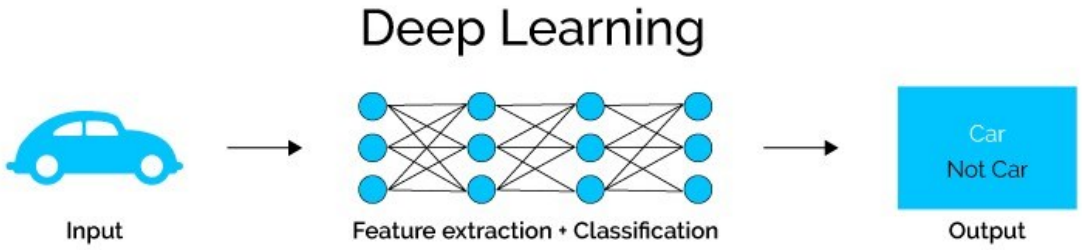


Figura 15 schema deep learning

Tra le architetture di apprendimento profondo si annoverano le reti neurali profonde, la convoluzione di reti neurali profonde, le Deep belief network, e reti neurali ricorsive, che sono state applicate nella visione artificiale, nel riconoscimento automatico del discorso, nell'elaborazione del linguaggio naturale, nel riconoscimento audio e nella bioinformatica. "Apprendimento profondo" è un'espressione oggi famosa che ridà lustro al concetto di rete neurale.

L'apprendimento profondo è definito come una classe di algoritmi di apprendimento automatico che:

- usano vari livelli di unità non lineari a cascata per svolgere compiti di estrazione di caratteristiche e di trasformazione. Ciascun livello successivo utilizza l'uscita del livello precedente come input. Gli algoritmi possono essere sia di tipo supervisionato sia non supervisionato e le applicazioni includono l'analisi di pattern (apprendimento non supervisionato) e classificazione (apprendimento supervisionato).
- sono basati sull'apprendimento non supervisionato di livelli gerarchici multipli di caratteristiche (e di rappresentazioni) dei dati. Le caratteristiche di più alto livello

vengono derivate da quelle di livello più basso per creare una rappresentazione gerarchica.

- fanno parte della più ampia classe di algoritmi di apprendimento della rappresentazione dei dati all'interno dell'apprendimento automatico.
- apprendono multipli livelli di rappresentazione che corrispondono a differenti livelli di astrazione; questi livelli formano una gerarchia di concetti.

Ciò che queste definizioni hanno in comune sono i livelli multipli di unità non lineari e l'apprendimento (supervisionato o non supervisionato) in ogni livello della rappresentazione di caratteristiche, in cui i livelli formano una gerarchia delle caratteristiche stesse. La composizione di ciascun livello di unità non lineari usata in un algoritmo di apprendimento profondo dipende dal problema che deve essere risolto. Nell'apprendimento profondo possono venire usati livelli nascosti di una rete neurale artificiale e insiemi di formule proposizionali.

Oggi i sistemi di apprendimento profondo, fra altre utilità, permettono di identificare oggetti nelle immagini e nei video; trascrivere il parlato in testo; individuare e interpretare gli interessi degli utenti online, mostrando i risultati più pertinenti per la loro ricerca.

Grazie a queste e altre soluzioni, l'apprendimento profondo sta vivendo anni di rapido progresso, arrivando anche, in molti casi, a superare le prestazioni degli esseri umani.

Clustering

La cluster analisi, o clustering, è in grado di rilevare similarità strutturali tra le osservazioni di un dataset attraverso l'assegnazione di un insieme di osservazioni in sottogruppi (cluster) di elementi tra loro omogenei. Il clustering è un metodo di apprendimento non supervisionato, e una tecnica comune per l'analisi statistica dei dati.

La cluster analysis è ad oggi uno degli strumenti più potenti e utilizzati tra quelli che pendono dalla cintura degli attrezzi di statistici e data scientist.

Il termine cluster, di per sé, sta a indicare un raggruppamento di oggetti che hanno uno o più caratteristiche in comune.

Se si prende come esempio un insieme di individui e di questi si conosce unicamente il colore degli occhi, è possibile suddividere il gruppo in un numero di cluster pari al numero di colori. Quindi si avrà un cluster contenente tutti gli individui con gli occhi azzurri, un altro contenente gli individui con gli occhi verdi e così via.

È possibile estendere questo ragionamento a un numero maggiore di attributi, immaginando per esempio che ogni individuo nel nostro insieme indossi una maglietta rossa o blu. Si può

quindi creare una nuova suddivisione in cluster per raggruppare tutti gli individui con gli occhi azzurri e la maglietta rossa, gli occhi azzurri e la maglietta blu, gli occhi verdi e la maglietta rossa e così via. Quindi, in sostanza, preso un insieme di oggetti aventi un certo numero di attributi, questi possono essere utilizzati per separare gli oggetti in un numero qualunque di cluster.

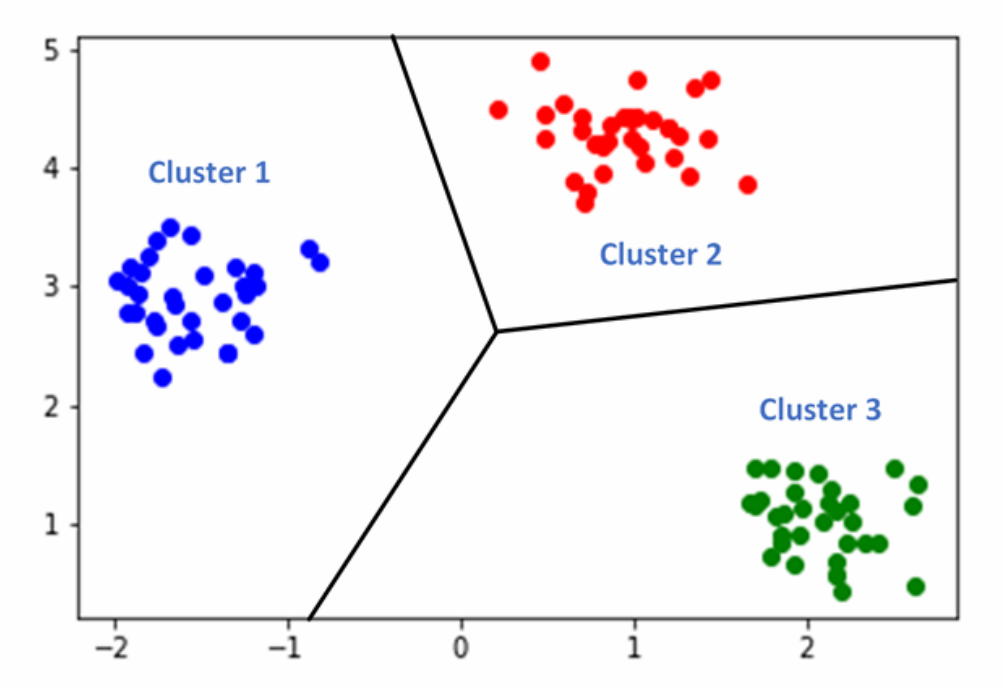


Figura 16 esempio di clustering

Definito in questo modo, il cluster assomiglia molto a un altro famoso membro della famiglia del Machine Learning: la classe. Anche una classe raccoglie al suo interno una serie di oggetti aventi caratteristiche in comune; la differenza sta principalmente nella tecnica utilizzata per definire di quale gruppo fa parte un determinato oggetto.

Quando si effettua una classificazione si hanno una serie di classi (categorie) note a priori e lo scopo è quello di capire a quale gruppo appartiene un oggetto osservando il valore dei suoi attributi. Per fare questo, durante la fase di addestramento di un artefatto (modello) di classificazione, si parte da un insieme di oggetti dei quali si conosce già la categoria di appartenenza. Attraverso l'analisi degli attributi degli oggetti appartenenti a una determinata classe si cerca di trovare un pattern comune. La classificazione è, quindi, un procedimento di apprendimento supervisionato dove la conoscenza di una determinata categoria esiste a prescindere dagli oggetti in essa raggruppabili.

Nella clusterizzazione, invece, si vuole estrapolare un certo numero di gruppi in cui è possibile separare gli oggetti di un insieme analizzando i valori dei loro attributi. In questo

caso non esistono classi predeterminate né esempi che le rappresentino. L'algoritmo deve riuscire a identificare gli oggetti che "si somigliano" e raggrupparli tra loro. Di conseguenza la clusterizzazione è un algoritmo di tipo non supervisionato.

Un algoritmo di clustering è, quindi, in grado di raggruppare tra loro oggetti che hanno caratteristiche simili. Come riesce a raggiungere questo scopo?

Esistono più tecniche che permettono di clusterizzare un insieme di oggetti. Una prima importante suddivisione dipende dalla tecnica di generazione dei cluster stessi, che divide gli algoritmi in due categorie:

- Algoritmi di clusterizzazione agglomerativi (bottom-up) Iniziano inserendo ogni oggetto dell'insieme in un proprio cluster per poi raggrupparli iterativamente fino al raggiungimento di una condizione specifica (es. numero di cluster desiderato).
- Algoritmi di clusterizzazione divisivi (top-down) Iniziano inserendo tutti gli oggetti dell'insieme in un unico cluster per poi separarlo iterativamente in cluster più piccoli fino al raggiungimento di una condizione specifica.

Il risultato finale, in entrambi i casi, è un insieme di cluster contenenti uno o più oggetti.

A prescindere dall'approccio utilizzato, gli algoritmi di clustering si basano tutti su una metrica, puramente geometrica, che permetta di identificare quanto simili siano due oggetti fra di loro. Infatti, gli oggetti in esame vengono visti come insiemi di valori reali che ne rappresentano le caratteristiche (colore degli occhi, altezza, peso, ecc.). Questi valori, a loro volta, possono essere raggruppati in modo da formare dei vettori che rappresentino punti in uno spazio euclideo.

Albero di decisione

Un albero di decisione è un grafo di decisioni e delle loro possibili conseguenze, (incluso i relativi costi, risorse e rischi) utilizzato per creare un 'piano' (plan) mirato ad uno scopo (goal). Esso è costruito al fine di supportare l'azione decisionale (decision making).

Nel machine learning un albero di decisione è un modello predittivo, dove il nodo principale si chiama root node o radice. Quando un nodo porta a una divisione in rami nei sottonodi l'operazione si chiama splitting. Quando un nodo si divide in più sottonodi senza arrivare a quello finale, i sottonodi si chiamano decision node o nodo di decisione. Ogni nodo figlio rappresenta un possibile valore per una proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà. Una intera sezione dell'albero si chiama branch o ramo.

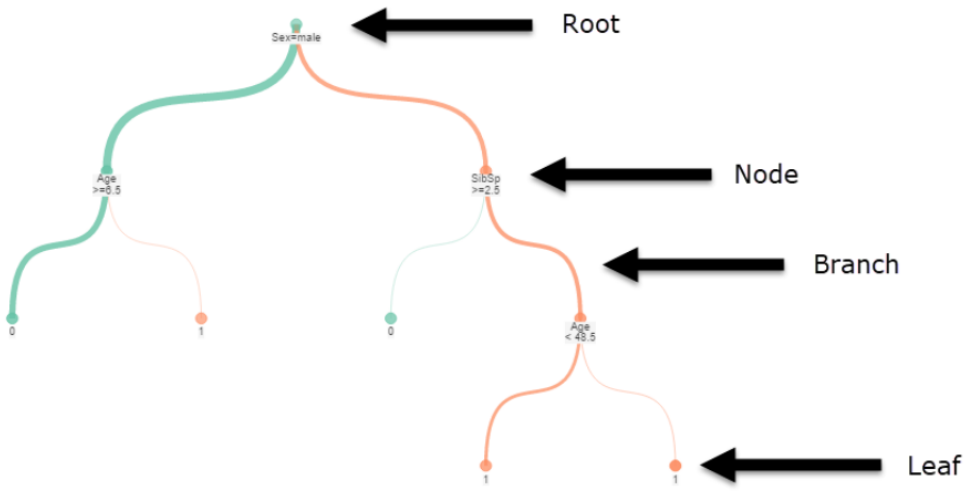


Figura 17 strtuttura albero decisionale

Normalmente un albero di decisione viene costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (dataset), il quale può essere diviso in due sottoinsiemi:

- il training set, sulla base del quale si crea la struttura dell'albero;
- il test set, che viene utilizzato per testare l'accuratezza del modello predittivo così creato.

Il predicato che si associa ad ogni nodo interno (sulla base del quale avviene la ripartizione dei dati) è chiamato condizione di split.

In molte situazioni è utile definire un criterio di arresto (halting), o anche criterio di potatura (pruning) al fine di determinarne la profondità massima. Questo perché il crescere della

profondità di un albero (ovvero della sua dimensione) non influisce direttamente sulla bontà del modello. Una crescita eccessiva della dimensione dell'albero, infatti, potrebbe portare solo ad un aumento sproporzionato della complessità computazionale rispetto ai benefici riguardanti l'accuratezza delle previsioni o classificazioni.

I parametri più largamente usati per le condizioni di split sono:

- tasso d'errore nella classificazione (misclassification error).
- indice di Gini (Gini index): utilizzato da CART (Classification and Regression Trees). L'indice di Gini raggiunge il suo minimo (zero) quando il nodo appartiene ad una singola categoria:

$$I_G(i) = 1 - \sum_{j=1}^{m} f(i,j)^2$$

- variazione di entropia (nota anche come entropy deviance): è basata sul concetto di entropia definito in teoria dell'informazione:

$$I_E(i) = -\sum_{j=1}^{m} f(i,j) \log f(i,j)$$

In entrambe le formule f rappresenta la frequenza del valore j nel nodo i.

L'indice di Gini e la variazione di entropia sono i parametri che vengono usualmente utilizzati per guidare la costruzione dell'albero, mentre la valutazione del tasso di errore nella classificazione viene utilizzato per effettuare una ottimizzazione dell'albero nota come processo di pruning ("potatura" dei nodi superflui). Poiché, in generale, in un buon albero di decisione i nodi foglia dovrebbero essere il più possibile puri (ovvero contenere solo istanze di dati che appartengono ad una sola classe), un'ottimizzazione dell'albero consiste nel cercare di minimizzare il livello di entropia man mano che si scende dalla radice verso le foglie. In tal senso, la valutazione dell'entropia determina quali sono, fra le varie scelte a disposizione, le condizioni di split ottimali per l'albero di classificazione.

Questo algoritmo è in grado di prendere un set di dati non familiari ed estrarre una serie di regole tramite le quali rendere l'uomo capace di comprendere il problema e i risultati, ed ha i seguenti vantaggi e svantaggi. Vantaggi:

- basso tempo di computazione
- facile comprensione dei risultati appresi
- può trattare anche features irrilevanti

Svantaggi:

- incline all'overfitting.

Per la costruzione dell'albero decisionale si parte fornendo all'algoritmo il dataset di punti contenenti tutti i valori associato ad ogni attributo, o feature, e i valori di target. I dati vengono suddivisi ogni volta nei nodi secondo i valori assunti dalle variabili features. Per determinare ciò, si prova ogni feature e si misura quale split fornisce i risultati migliori. Dopo di questo, viene diviso il dataset in sottoinsiemi, i quali sono fatti passare attraverso i primi rami del primo nodo decisionale: se i dati analizzati rispettano la stessa condizione imposta alla feature viene raggiunto il nodo foglia, altrimenti si prosegue verso un altro nodo decisionale in cui vengono divisi secondo la condizione nel nuovo nodo. Ogni nodo foglia finale definisce una certa regione, entro le quali saranno poi valutate le nuove variabili obiettivo. Un esempio di un decision tree per la regressione è rappresentato in figura.

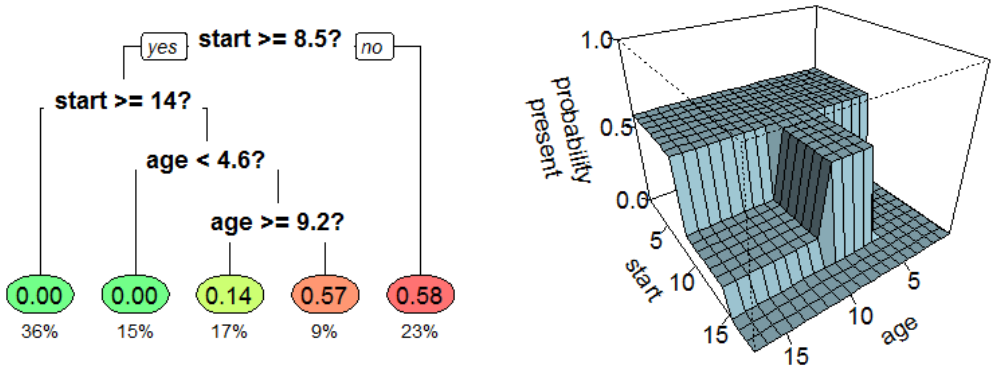


Figura 18 rappresentazioni di decision tree

I Decision Tree adottano veramente poche assunzioni rispetto al training dataset (diversamente dai modelli lineari, per esempio, che ovviamente assumono dei dati lineari). Pertanto, se non vincolata, la struttura dell'albero si adatterà ai dati di training nel modo migliore possibile, cadendo molto probabilmente in overfitting. A tal proposito, si distinguono i modelli in:

- Nonparametric: i molti parametri disponibili non sono determinati a priori sul training, così che la struttura è libera di adattarsi perfettamente ai dati;
- Parametric: possiede un numero di parametri predeterminati, così i gradi di libertà sono limitati e, conseguentemente, viene ridotto il rischio di overfitting. Come è noto a questo punto, per evitare questo problema è necessario limitare la libertà del decision tree durante l'allenamento. Tale processo è chiamato regolarizzazione degli iperparametri e, nonostante dipenda dall'algoritmo utilizzato, si può almeno limitare la massima profondità dell'albero. In Scikit-Learn, i principali iperparametri che controllano la crescita della struttura e della

forma del Decision Tree sono:

- max_depth: indica la massima profondità del Decision Tree, definita in livelli;
- min_samples_split: numero minimo di istanze che un nodo deve avere prima di essere diviso;
- min_samples_leaf: numero minimo di istanze che un nodo foglia deve avere;
- max_features: massimo numero di features valutate per dividere ogni nodo.

Incrementando quelli che esprimono un valore minimo o diminuendo quelli che esprimono un valore massimo si può regolarizzare il modello.

Risulta chiaro, quindi, come i Decision Trees siano semplici da comprendere e interpretare, facili da utilizzare, versatili e potenti. Tuttavia, hanno delle limitazioni, alcune già citate. Infatti, impostando dei valori di features nei nodi, realizzano delle condizioni al bordo ortogonali, visibile anche in Figura, e ciò li rende del tutto sensibili a piccole variazioni del training set, come per esempio una sua rotazione. Pertanto, dal momento che l'algoritmo di training usato da Scikit-Learn è stocastico, si possono avere modelli diversi anche lavorando sullo stesso dataset di training (a meno che non si imposti un valore all'iperparametro random_state). Tale problema viene chiamato instability, e può essere limitato fortemente dal famoso algoritmo Random Forest, il quale fa una media tra le predizioni su tanti alberi, come vedremo nel prossimo paragrafo.

Ensemble Learning e Random Forest

Quando si vuole ottenere la predizione di un certo evento, risulta più efficace l'utilizzo di un insieme di predittori piuttosto che il miglior predittore singolo, ottenendo quasi sempre previsioni migliori. Il gruppo di predittori viene chiamato ensemble e, di conseguenza, questa tecnica è denominata Ensemble Learning. Un algoritmo che sfrutta questa metodologia è chiamato Ensemble method. Un gruppo di Decision Trees, ognuno con un diverso sottoinsieme casuale del training set, che forniscono una previsione finale è chiamato Random Forest ed è uno dei più potenti algoritmi di Machine Learning, nonostante la sua semplicità.

Come già accennato, per ottenere una buona previsione finale occorre avere una serie di predittori che usano diversi algoritmi di training. Un altro modo altrettanto efficace è quello di utilizzare lo stesso algoritmo di apprendimento per ogni predittore, ma quest'ultimi allenati su diversi sottoinsiemi del training set. Si può distinguere l'assegnazione dei sottoinsiemi ai singoli predittori in due categorie:

- Bagging (bootstrap aggregating): con ricambio;
- Pasting: senza ricambio.

Entrambi permettono ai dati di training di essere selezionati molte volte attraverso molti predittori. Il processo di selezione è mostrato nella Figura.

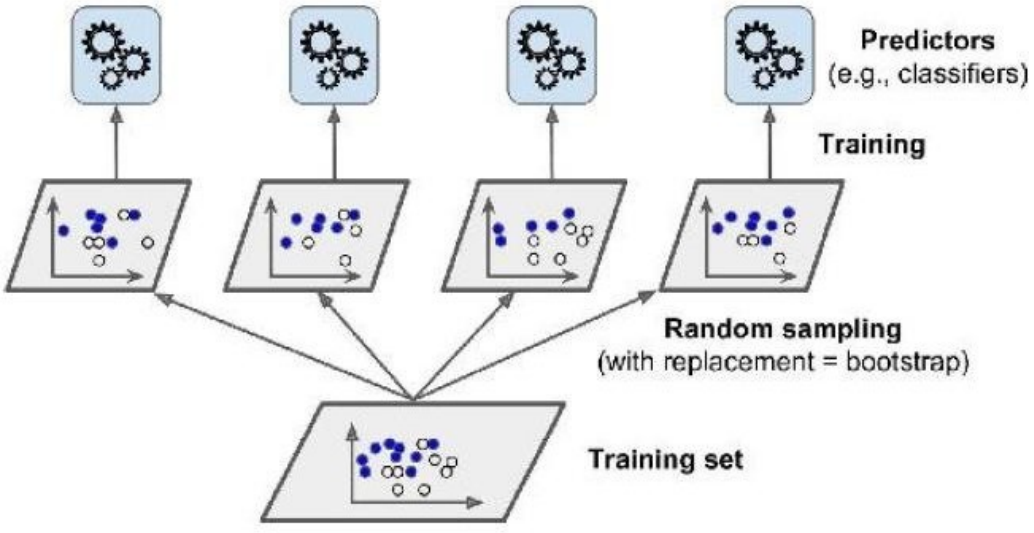


Figura 19 processo di selezione

Una volta che tutti i predittori sono stati allenati, l'ensemble può fornire la previsione per un nuovo dato semplicemente aggregando quella di tutti i predittori stessi. Come si può notare, i predittori sono allenati tutti in parallelo e ciò permette alla macchina di usare diversi core della CPU; anche la predizione può essere fatta in parallelo.

Ovviamente, come ci possiamo aspettare, la previsione di un gruppo di predittori risulterà molto più generalizzante rispetto al singolo, evitando così l'overfitting: l'ensemble possiede una varianza più piccola.

L'algoritmo Random Forest è, quindi, un insieme di Decision Trees generalmente allenati con il metodo di bagging (o qualche volta pasting) e può essere utilizzato per problemi discreti, classificazione, o continui, regressione, quest'ultimo costituente il presente caso. Questo algoritmo aggiunge un'ulteriore casualità, oltre all'assegnazione dei sottoinsiemi del training set ai singoli predittori: invece di ricercare la migliore feature come condizione per suddividere un nodo, trova la migliore feature in un sottoinsieme casuale dell'insieme globale delle feature stesse. Questo conduce ad una maggior diversità dei singoli alberi decisionali.

Una caratteristica fondamentale e di grande utilità, appartenente al Random Forest, è costituita dalla Feature Importance: funzione tramite la quale questo tipo di algoritmi misurano facilmente l'importanza relativa di ogni feature. Scikit-Learn trova l'importanza di una singola feature semplicemente calcolando quanto i nodi dell'albero, che utilizzano tale feature, riducono le impurità sulla media di tutti gli alberi decisionali. In altre parole, è una media pesata dove ogni peso di un nodo è uguale al numero di dati di allenamento associati al nodo stesso. Questa tecnica viene attuata successivamente all'allenamento, e i valori di importanza associati alle features sono tali che la loro somma sia uguale a uno.

Training e Valutazione Prestazioni

In genere, il comportamento di un algoritmo di Machine Learning è regolato da un set di parametri che lo caratterizzano. L'apprendimento consiste nel determinare il valore ottimo di questi stessi parametri.

Quindi, dato un training set e un insieme di parametri, la funzione obiettivo può indicare:

- L'ottimalità della soluzione da massimizzare;
- L' errore o perdita (detta loss-function) da minimizzare.

Tale funzione può essere ottimizzata esplicitamente, con metodi che operano a partire dalla sua definizione matematica (come, per esempio, il calcolo alle derivate parziali), oppure implicitamente, utilizzando metodi euristici che modificano i parametri in modo coerente con la funzione stessa.

La maggior parte degli algoritmi richiede di definire, prima dell'apprendimento vero e proprio, il valore dei cosiddetti iperparametri e, una volta scelti opportunamente, si esegue l'apprendimento per ciascuno, andando a prendere quelli che hanno fornito le prestazioni migliori. Esempi di iperparametri possono essere il numero di neuroni in una rete neurale, il grado di un polinomio usato in una regressione, il tipo di loss-function, il numero di alberi e foglio in un algoritmo Random Forest, e così via.

Andiamo allora a dare una definizione di quelle che vengono considerate le prestazioni dell'algoritmo e a definire i parametri che le caratterizzano.

Valutazione delle Prestazioni

Le prestazioni di un algoritmo possono essere valutate utilizzando direttamente la funzione obiettivo per quantificare le prestazioni. In genere, però, è preferibile usare una misura direttamente legata alla semantica del problema, ovvero parametri matematici del campo della statistica e probabilità. Anche in questo caso, si può fare una distinzione dipendentemente dal tipo di sistema, cioè discreto o continuo.

Discreto: in un problema di Classificazione, l'accuratezza (da 0% a 100%) è la percentuale di pattern correttamente classificati, mentre l'errore è il complemento di questa.

$$Accuratezza = \frac{pattern\ correttamente\ classificati}{pattern\ classificati}$$

$$Errore = 100\% - Accuratezza$$

Continuo: in un problema di regressione viene valutato il root mean square error (rmse), cioè la radice della media dei quadrati degli scostamenti tra il valore vero e il valore predetto. In poche parole, questo parametro fornisce un'idea di quanto il sistema, con la sua predizione, si allontana dalla realtà dei dati ed è espresso dall'equazione:

$$rmse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(pred_i - true_i)^2}$$

Anche se rmse è generalmente preferibile come misura delle prestazioni in un algoritmo di regressione, talvolta si può utilizzare un'altra funzione chiamata mean absolute error (mae):

$$mae = \frac{1}{N}\sum_{i=1}^{N}|pred_i - true_i|$$

Quindi, sia rmse che mae sono due modi per misurare la distanza tra due vettori, quello delle predizioni e quello dei valori di target.

Training, Validation and Test

Il miglior modo per sapere come un modello si adatterà ai nuovi dati è provarlo sulle istanze stesse. Per far ciò il dataset fornito all'algoritmo viene diviso in due parti: training e testing, a sua volta il primo viene suddiviso in training set e validation set.

Training test: insieme di pattern tramite il quale addestrare l'algoritmo trovando il valore ottimo dei parametri.

Validation test: insieme di pattern su cui il sistema tara gli iperparametri.

Testing set: insieme di pattern su cui l'algoritmo valuta le prestazioni finali.

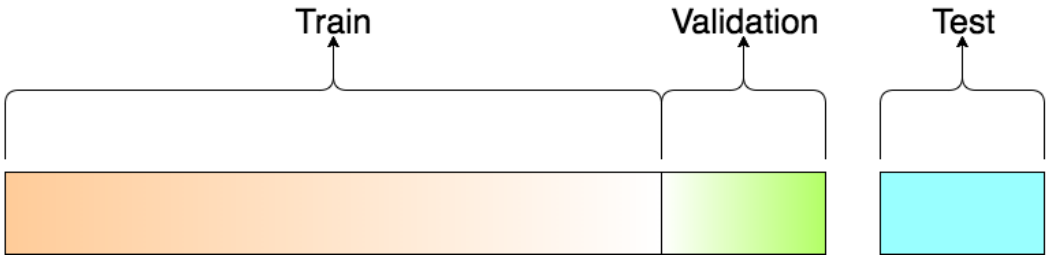


Figura 20 train set, validation set, test set

Per evitare di utilizzare troppi dati di training nel validation set e averne, quindi, a sufficienza per l'allenamento dell'algoritmo, una comune tecnica è quella della cross-

validation: l'insieme dei dati di training viene suddiviso in sottoinsiemi complementari, dopo di che il modello è allenato su una parte di questi e validato sulla rimanente. Tale operazione è ripetuta per diverse combinazioni dei sottoinsiemi. Una volta definita quella che restituisce le prestazioni migliori e selezionati gli iperparametri, il modello finale viene allenato su il training set completo e, successivamente, applicato al test set per fornire l'errore generalizzato. Valutando l'algoritmo sul test set si ha una misura del tasso di errore della predizione rispetto al caso di target. Tale è chiamato proprio errore generalizzato, o out-of-sample error.

Uno dei primi obiettivi da perseguire durante l'addestramento è la convergenza sul train set. Se consideriamo un classificatore in cui l'addestramento prevede un processo iterativo, si ottiene convergenza quando:

- l'output della loss-function ha andamento decrescente rispetto al numero di iterazioni;
- l'accuratezza ha andamento crescente rispetto al numero di iterazioni.

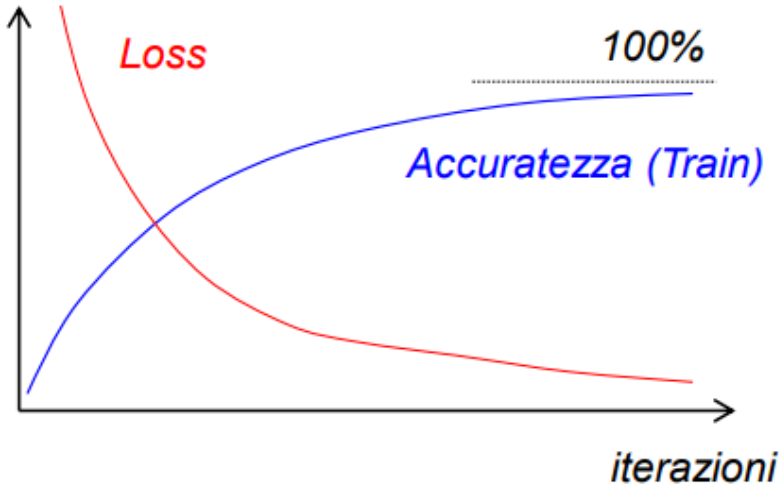


Figura 21 confronto tra loss e accuracy

Se gli andamenti non sono quelli mostrati in figura significa che molto probabilmente saranno presenti altre situazioni, vengono riportati alcuni esempi.

Se il loss non decresce (o oscilla) il sistema non converge: il metodo di ottimizzazione non è efficace, gli iperparametri sono fuori range, il tasso di apprendimento è inadeguato, possono esserci errori di implementazione, e così via.

Se il loss decresce ma l'accuratezza non cresce, probabilmente è stata scelta una loss-function errata.

Se l'accuratezza non si avvicina al 100% sul train, i gradi di libertà del classificatore non sono sufficienti per gestire la complessità del problema.
Di seguito sono brevemente spiegati i due errori più comuni e importanti che possono verificarsi durante l'addestramento del modello: overfitting e underfitting.

Overfitting e Underfitting del Training Data

L'obiettivo ultimo di un algoritmo di Machine Learning è quello di massimizzare l'accuratezza sul test set. Se si ipotizza che i risultati ottenuti sul validation set siano rappresentativi del testing, allora dobbiamo massimizzare l'accuratezza della parte di validazione. Un modello, per far sì che sia ben costruito, allenato e validato deve avere una generalizzazione adatta al tipo di problema.
Dopo l'allenamento, l'algoritmo di apprendimento raggiungerà uno stato in cui sarà in grado di predire gli output per i pattern ancora non visionati (fase di test), cioè sarà in grado di generalizzare.
Tuttavia, soprattutto nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove c'era uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi.
Quando un modello ha un numero eccessivo di gradi di libertà, o parametri che lo caratterizzano, rispetto al numero di osservazioni, raggiunge un elevata accuratezza sul train set ma non sulla validazione (scarsa generalizzazione): si parla di overfitting di train.

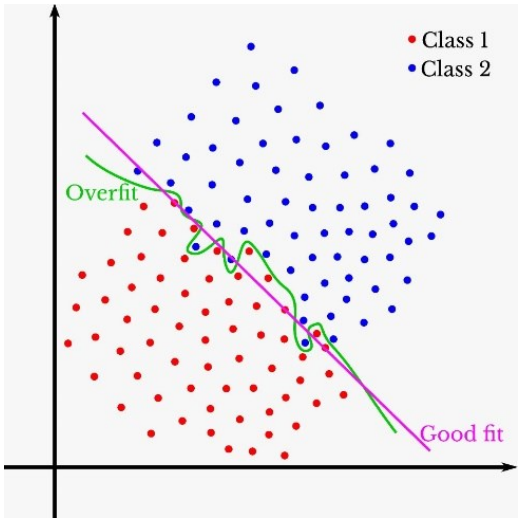


Figura 22 esempio di overfitting

Risulta quindi buona norma partire con pochi gradi di libertà, controllabili attraverso gli iperparametri, e via via aumentarli monitorando accuratezza su train e validation set.

Opposto all'overfitting abbiamo l'underfitting, il quale si verifica quando il modello è troppo semplice per apprendere dalla struttura dei dati forniti. Questo può accadere quando i parametri che definiscono l'algoritmo sono troppo pochi rispetto alla mole di dati di apprendimento. Le principali strategie per risolvere tale problema possono essere:

- selezionare un modello più potente, con più parametri;
- fornire migliori features all'algoritmo;
- ridurre i vincoli del modello.

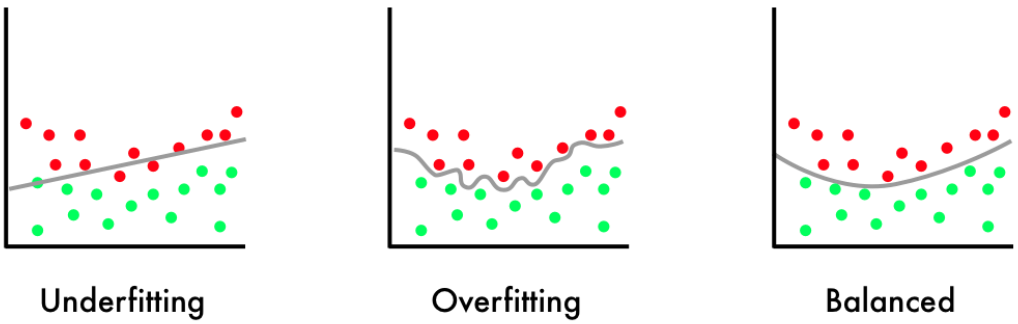


Figura 23 confronto tra underfitting, overfitting e bilanciato

OEE

L'OEE, (Overall Equipment Effectiveness) letteralmente "efficienza generale dell'impianto", è una grandezza adimensionale che può assumere valori tra 0 e 1, ma che viene più spesso espressa in punti percentuali. Esso riassume in sé tre diversi aspetti molto importanti dal punto di vista del controllo di produzione: la disponibilità, l'efficienza e la qualità del processo.

Esso rappresenta quindi la percentuale di tempo di produzione che è realmente produttiva; per questo un OEE del 100% (o pari a 1) significa che vengono prodotti solo pezzi buoni (ossia con qualità del 100%), alla massima velocità (le prestazioni sono al 100%) e senza interruzioni (le macchine sono disponibili per il 100% del tempo). È però impensabile che qualsiasi processo di produzione possa essere eseguito al 100% di OEE. I produttori che decidono di fissare per le loro fabbriche obiettivi impegnativi solitamente non si spingono oltre l'85%. L'OEE può essere anche definito, in altre parole, come il rendimento globale di una risorsa produttiva o di un insieme di risorse, siano esse umane o tecniche, durante i periodi in cui è pianificata l'esecuzione.

La forma generica dell'OEE consente il confronto tra unità produttive in diversi settori.

Per capire meglio come calcolarlo bisogna prima fare una distinzione tra efficienza ed efficacia (o rendimento), dato che molte volte questi due termini tendono ad essere usati come sinonimi quando invece sono profondamente diversi.

Per efficienza si intende la capacità produttiva rispetto a quella teorica, per efficacia (o rendimento) si intende invece la capacità di produrre pezzi conformi rispetto alla capacità teorica.

Questo porta alla conclusione che l'efficacia (o rendimento) è minore dell'efficienza, in quanto è impossibile avere un impianto con un rendimento superiore alla sua efficienza.

L'OEE si occupa di riunire tutti questi concetti sotto un unico valore, per avere quindi un'efficacia generale dell'impianto. È l'indicatore più "esigente" ed omnicomprensivo che esista, in quanto tiene conto di tutte le tipologie di inefficienze che portano ad una minore produttività: dalla mancanza di materiali alla cattiva pianificazione, dai setup ai tempi morti, dalle micro-fermate ai guasti, dalle rilavorazioni alle non conformità. L'OEE non è tuttavia una misura assoluta ed è meglio utilizzarlo solamente per ottenere spunti importanti su come migliorare sistematicamente il processo di produzione e la produttività delle attrezzature, eliminando gli sprechi.

Calcolo OEE

Vediamo ora come può essere calcolato questo indicatore.

Il modo più semplice per ottenere l'OEE è quello di dividere il tempo minimo necessario per produrre le parti in condizioni ottimali per il tempo effettivo necessario per produrre le parti. Per esempio, considerando la durata di un turno di 8 ore (480 minuti) con una produzione di 240 pezzi per turno, si ha un tempo effettivo per pezzo di 2 minuti. Il tempo di ciclo più veloce possibile però è di 1,5 minuti, quindi sarebbero stati necessari solo 360 minuti per produrre tutti i 240 pezzi. I restanti 120 minuti sono andati persi. L'OEE è quindi pari a 360 minuti divisi per 480 minuti (pari a un minimo di 1,5 minuti per pezzo diviso per 2 minuti effettivi), ovvero 0,75 (75%).

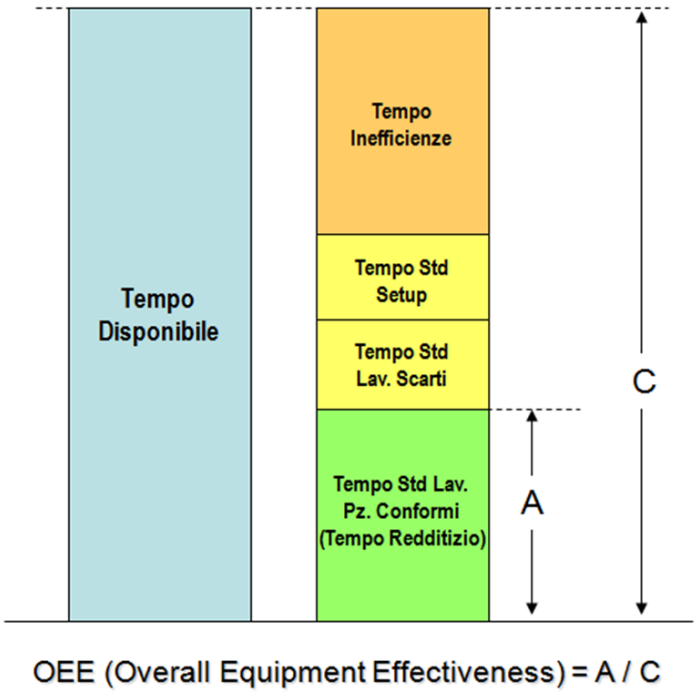


Figura 24 suddivisione del tempo disponibile

Il secondo metodo è quello di calcolare l'OEE come prodotto di tre componenti distinti:

$$OEE=D\cdot E\cdot Q$$

Dove D sta per tasso di disponibilità, E per efficienza di lavorazione e Q per tasso di qualità. Questo metodo è anche il più utile, perché permette di rendersi conto separatamente di ognuna delle tre variabili, che può quindi diventare un target di miglioramento del processo produttivo.

Andiamo ad analizzare più nel dettaglio ogni parametro.

Il tasso di disponibilità è il rapporto tra il Tempo Operativo e il Tempo Disponibile per la produzione. Viene anche indicata con il termine Available Time o Scheduled Time.
Per tempo disponibile per la produzione si intende il tempo totale disponibile della macchina (quindi tutto l'anno) meno le chiusure pianificate (quindi le manutenzioni programmate, le manutenzioni preventive, mancanza di ordini, chiusure aziendali, ecc.)
Per tempo operativo si intende il tempo disponibile per la produzione sottratto del tempo in cui la macchina è occupata per attività non pianificate. Le perdite per inattività sono misurate in unità di tempo. Esse includono: guasti e tempi di riparazione, tempi di set-up e regolazione, altre perdite causa di inattività. Se chiamiamo UPTIME il tempo operativo e DOWNTIME il tempo di fermo, otterremo che:

$$D = \frac{\text{UPTIME}}{\text{UPTIME} + \text{DOWNTIME}}$$

L'efficienza di lavorazione o produttività è il rapporto tra Tempo Operativo Netto e il Tempo Operativo.
Il Tempo Operativo è sempre il Tempo Disponibile per la Produzione meno il tempo in cui la macchina è occupata per attività non pianificate.
Il Tempo Operativo Netto è il Tempo Operativo meno la quantità di tempo persa a causa di inefficienze produttive. Ogni processo ha una propria efficienza e nel caso di produzione essa rappresenta il rapporto tra la cadenza reale della macchina e quella teorica. Queste due cadenze dovrebbero essere uguali, ma molto spesso le macchine, per svariati motivi, sono più lente di quanto programmato, cioè sono soggette a microfermate, non ascrivibili nel Downtime perché difficilmente misurabili, che determinano l'inefficienza.
Proprio perché sarebbe difficile tenere traccia delle microfermate, l'efficienza è una variabile che si calcola deduttivamente, attraverso il rapporto tra il numero di pezzi effettivamente prodotti in un intervallo di tempo di UPTIME ed il numero teorico programmato.

$$E = \frac{\text{Numero pezzi effettivamente prodotti}}{\text{Numero pezzi teoricamente da produrre}}$$

Si può anche pensare all'efficienza di lavorazione come la velocità con cui l'impianto sta lavorando rapportata alla velocità di progetto. Le perdite dovute alla produttività sono spesso chiamate anche perdite di velocità. In pratica è spesso difficile determinare le perdite di velocità e un approccio comune è semplicemente assegnare le perdite sconosciute rimanenti come perdite di velocità.

Il tasso di qualità è il rapporto tra Tempo Operativo a Valore e Tempo Operativo Netto.
Il Tempo Operativo Netto è quello già calcolato per l'efficienza di lavorazione.
Il Tempo operativo a Valore è il Tempo Operativo Netto sottratto del tempo perso per difetti di qualità. (scarti, tempi di startup, rilavorazioni).
Le unità che possono essere rielaborate e che quindi verranno corrette vengono conteggiate solo come tempi di inattività non programmati mentre le unità da buttare influenzano sia il tempo di funzionamento che il conteggio delle unità.
Il tasso di qualità può anche essere misurato come rapporto tra pezzi realizzati conformi alle specifiche e il numero totale di pezzi effettivamente prodotti.

$$Q=\frac{\text{Numero pezzi realizzati conformi}}{\text{Numero pezzi realizzati effettivamente}}$$

Metodo OEE

Raccogliendo i dati dell'OEE su base fissa, è possibile individuare i procedimenti e le interferenze che causano problemi all'attrezzatura produttiva. Inoltre, i dati raccolti permettono di valutare se gli interventi messi in atto per migliorare le prestazioni delle macchine hanno dato risultati positivi. Affinché il processo di misurazione e applicazione dei dati OEE risulti efficace dovrebbe essere coinvolto il personale operativo, il quale inoltre dovrebbe ricevere feedback (informazioni di ritorno) sui risultati dell'OEE.

Il miglioramento continuo dell'OEE è l'obiettivo del TPM (Total Productive Maintenance), un sistema produttivo che mira al raggiungimento della massima efficienza aziendale. Storicamente nasce per garantire la massima efficienza dei singoli impianti, focalizzando l'attenzione sulle attività degli operatori, dei manutentori e dei tecnici di processo. Successivamente vengono strutturate anche le attività che riguardano la qualità, lo sviluppo del personale, le attività di sicurezza e ambiente e di industrializzazione. Per raggiungere questo obiettivo, il TPM definisce una tattica di miglioramento che punta alla riduzione di ciascuno dei tipi di perdite di OEE, dotandosi di un efficace sistema di misurazione e controllo.

L'attuale contesto competitivo è caratterizzato da una progressiva riduzione dei margini, dovuta alla crescente globalizzazione e ad una minore disponibilità alla spesa da parte dei clienti.

Questo impone alle aziende di ridurre i costi produttivi per conservare o migliorare la propria competitività. Una chiave essenziale per ottenere questo risultato è massimizzare l'efficienza delle risorse produttive, siano esse reparti, linee o centri di servizio.

Essere più efficienti significa, concretamente, aumentare la produttività a parità di risorse impiegate. Questo significa che si può ottenere la stessa produttività impiegando meno risorse, oppure ottenere più produttività dalle risorse esistenti. Spesso si è portati a pensare che il modo migliore per aumentare l'efficienza produttiva sia quello di rinnovare o potenziare gli asset tecnologici, affrontando investimenti spesso non trascurabili. L'esperienza però insegna che, in assenza di un adeguato sistema di controllo e miglioramento dell'efficienza, difficilmente un sistema produttivo esprime più del 50-60% del proprio potenziale.

Concepita inizialmente per la produzione a flusso continuo, questa tecnica può essere implementata con successo anche in contesti produttivi per celle o per reparti, con

possibilità di applicazione anche a centri di servizio che svolgono mansioni tendenzialmente ripetitive e standardizzate.

Il metodo OEE prevede alcuni step fondamentali:

- definizione/validazione degli standard (tempi e metodi)
- implementazione del sistema di registrazione delle inefficienze
- sviluppo dello strumento per il calcolo dell'efficienza e per l'analisi delle inefficienze
- formazione al personale sugli obiettivi dell'attività e sul significato degli indicatori
- valutazione del livello attuale di efficienza
- individuazione delle principali fonti di inefficienza
- eliminazione delle inefficienze tramite attività di miglioramento mirate
- quantificazione del miglioramento ottenuto e feedback

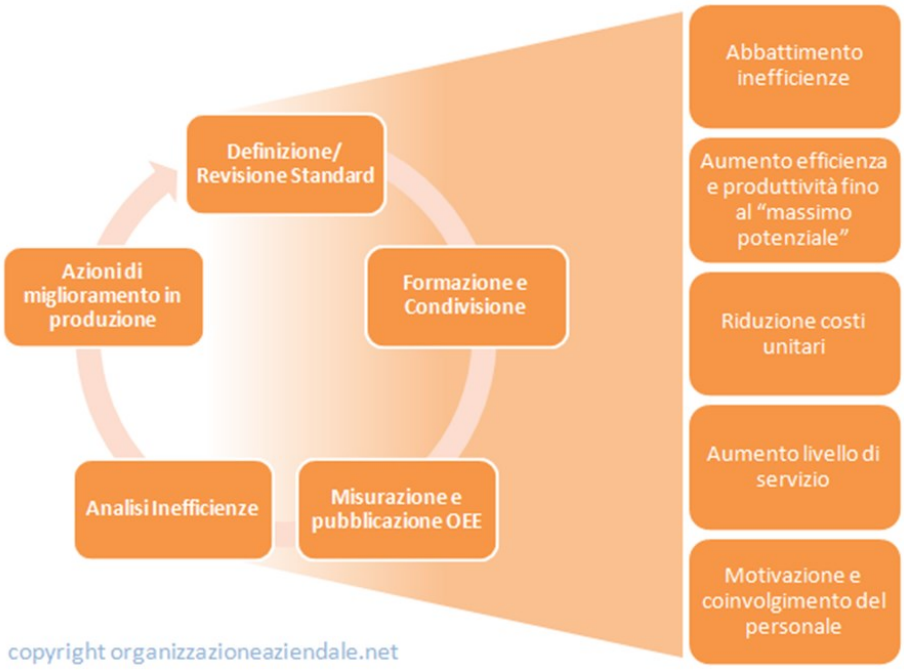


Figura 25 sintesi metodo OEE

La tecnica viene solitamente implementata in un'area pilota (reparto, linea produttiva o ufficio) e quindi progressivamente estesa alle altre aree. Ciò consente all'azienda di impostare un percorso sostenibile in base al contesto.

La pubblicazione degli indicatori (OEE %, incidenza degli attrezzaggi, analisi delle inefficienze etc.) costituisce un riferimento condiviso per tutto il personale coinvolto.

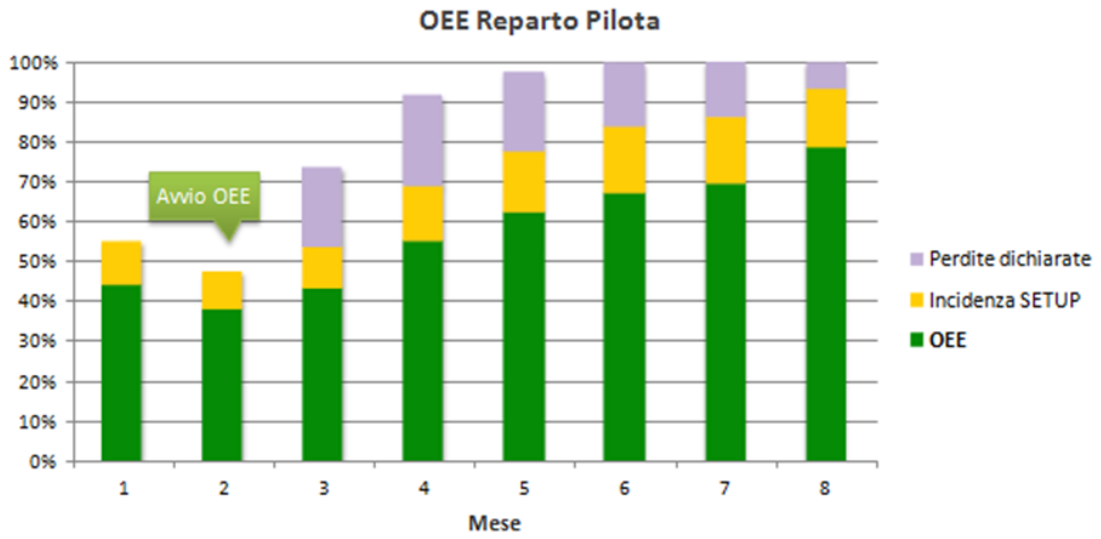


Figura 26 esempio miglioramento apportato

Tipicamente, un progetto OEE consente in pochi mesi di elevare l'efficienza produttiva anche del 30-40%, a seconda della situazione di partenza, portandola alla sua massima potenzialità. Le inefficienze individuate si rivelano spesso di natura prevalentemente organizzativa, e pertanto la loro risoluzione richiede investimenti molto bassi o addirittura nulli.

Di conseguenza, l'implementazione di un progetto OEE è solitamente caratterizzato da un ritorno dell'investimento molto rapido, oltre a garantire un guadagno di efficienza significativo e stabile nel tempo.

Case study: Magneti Marelli

Magneti Marelli è una multinazionale italiana specializzata nella fornitura di prodotti e sistemi ad alta tecnologia per l'industria automobilistica con sede a Corbetta (MI), Italia. Con 8,2 miliardi di euro di fatturato nel 2017, circa 44.000 addetti, 85 unità produttive, 15 centri R&D, il gruppo è presente in modo capillare in 20 Paesi (Italia, Francia, Germania, Spagna, Regno Unito, Polonia, Repubblica Ceca, Romania, Russia, Serbia, Slovacchia, Turchia, USA, Messico, Brasile, Argentina, Cina, Giappone, India e Malesia). Magneti Marelli fornisce tutti i maggiori car makers in Europa, Nord e Sud America e nella regione Asia Pacifico. Magneti Marelli punta a valorizzare, attraverso un processo di innovazione continua, il know-how e le competenze trasversali al fine di sviluppare sistemi e soluzioni che contribuiscano all'evoluzione della mobilità secondo criteri di sostenibilità ambientale, sicurezza e qualità della vita all'interno dei veicoli. Nella sua missione di componentista automotive a livello globale, Magneti Marelli mira a coniugare qualità e offerta competitiva, tecnologia e flessibilità, con l'obiettivo di rendere disponibili prodotti d'eccellenza a costi competitivi. Magneti Marelli opera a livello internazionale attraverso le seguenti aree di business:

- Automotive Lighting (sistemi di illuminazione)
- Electronic Systems (quadri di bordo, infotainment & telematica, lighting & body electronics)
- Powertrain (sistemi controllo motore benzina, diesel e multifuel, sistemi ibrido-elettrici e componenti, trasmissioni)
- Suspension Systems (sistemi sospensioni, ammortizzatori, dynamic system – sistemi di controllo dinamico del veicolo)
- Exhaust systems (sistemi di scarico, convertitori catalitici, silenziatori)
- Motorsport (sistemi elettronici ed elettromeccanici specifici per le competizioni con leadership tecnologica in Formula1, MotoGP, World SBK e WRC)
- Aftermarket Parts and Services (distribuzione ricambi per l'Independent Aftermarket – IAM; Rete Assistenza e Officine Checkstar)

La storia

Magneti Marelli nasce a Sesto San Giovanni nel 1919, con capitale sociale diviso in parti uguali tra Fiat e la società Ercole Marelli. La produzione iniziale riguarda magneti e componentistica per le auto, ma presto si estende agli equipaggiamenti elettrici, candele e batterie. Fin dalla nascita, Magneti Marelli è presente nelle competizioni sportive - per le quali fornisce tecnologia e assistenza su tutti i campi di gara, dalle auto alle moto, dalla nautica all'aereonautica – e nel mondo delle officine e dei ricambi. Nel corso degli anni Magneti Marelli ha raccolto l'eredità industriale di storici e conosciuti marchi europei in ambito automotive, come Carello e Siem nel campo dell'illuminazione, Veglia Borletti e Jaeger nel campo dei quadri strumenti, Weber e Solex nel campo dell'alimentazione e controllo motore. Non solo automotive, ma anche telecomunicazioni: nel 1930 Magneti Marelli avvia la produzione di massa di radio con il marchio Radiomarelli, e nel 1939 sperimenta le prime trasmissioni televisive, quasi vent'anni prima del loro lancio ufficiale. Tra gli anni '50 e '60 Magneti Marelli rende possibile l'avvio delle prime trasmissioni televisive della RAI, Radio Televisione Italiana, sviluppando e producendo macchine da presa, trasmettitori, ricevitori, i ponti radio per il primo e il secondo canale, oltre che i famosi televisori Radiomarelli. Nel 1967 l'intero capitale sociale viene rilevato da Fiat e fra gli anni '70 e '80 Magneti Marelli focalizza la sua attività esclusivamente nell'ambito automotive, trasformandosi in una holding industriale. Con la costituzione di Marelli Autronica, Magneti Marelli assume un ruolo di primo piano nello studio e nella produzione di dispositivi di controllo elettronico dei sistemi di accensione e alimentazione. Su questa linea, negli anni '90 Magneti Marelli assume un ruolo di riferimento nell'ambito dell'elettronica – che riveste un'importanza sempre maggiore nel funzionamento del sistema-automobile – iniziando anche le sue attività nel campo della navigazione e dei dispositivi di infotainment. Nel 2001 Magneti Marelli assume il controllo completo di Automotive Lighting, joint venture avviata con Bosch nel 1999. Dopo il 2000 Magneti Marelli rafforza la propria presenza nei settori illuminazione, powertrain e sistemi elettronici e nei principali mercati mondiali come Brasile, Cina e India, ampliando la sua vocazione di grande componentista con la capacità di progettare e realizzare sistemi e componenti automotive ad alta tecnologia per tutti i migliori car maker al mondo. Sul fronte tecnologico, dal 2000 in poi Magneti Marelli sviluppa soluzioni per dare le migliori risposte alle sfide poste dall'evoluzione dell'automotive. Dopo il 2010, mutuando anche l'esperienza nel motorsport, Magneti Marelli inizia a sviluppare soluzioni per la

propulsione ibrida ed elettrica, accanto a un forte focus sulle tecnologie di iniezione diretta ad alta pressione GDI. Nell'ambito dei sistemi elettronici e dell'interfaccia uomo-macchina (HMI) evoluta, Magneti Marelli nell'ultimo decennio propone un'offerta evoluta in termini di quadri di bordo digitali riconfigurabili e display con tecnologia OLED. In ambito motorsport, per la Formula 1 in particolare, tra il 2008 e il 2009 Magneti Marelli sviluppa il sistema KERS per il recupero dell'energia cinetica in frenata. Inoltre, dal 2016 fornisce a tutte le monoposto il sistema di telemetria V2X ad alta capacità, in grado di trasmettere in tempo reale e contemporaneamente, dalla macchina ai box, segnali audio ad alta qualità e a un grande volume di dati di telemetria. Da 2013 Magneti Marelli è fornitore ufficiale di Dorna per il sistema di controllo elettronico unico per tutta la MotoGP e dalla stagione 2019 fornirà il Sistema di Controllo Elettronico per la classe Moto2. Tra il 2017 e il 2018, le tappe storiche rilevanti più recenti sono: l'accordo con lo Stato del Marocco per la realizzazione di un insediamento produttivo a Tangeri, previsto nel 2019; la partecipazione societaria in LeddarTech, leader mondiale nella tecnologia di rilevamento LiDAR (Light Detection And Ranging) a stato solido; l'acquisizione di SmartMeUp, start-up specializzata nello sviluppo di software per l'elaborazione di segnali provenienti da sensori, in ambito guida autonoma.

Linea di assemblaggio

È stata analizzata la Main Assembly line (linea di montaggio) e EOL test and packaging (test ed imballaggio) della GDI Pump (Gasoline Direct Injection Pump).

Figura 27 pompa PHP

La Magneti Marelli PHP è una pompa ad un solo pistone azionata meccanicamente da camma multilobata (2, 3 e 4 lobi) e con controllo della portata tramite attuatore elettromagnetico.

Il pistone con diametro di 10 mm garantisce l'erogazione di portate elevate, di conseguenza la pompa è adatta anche per motori di elevata potenza e cilindrata.

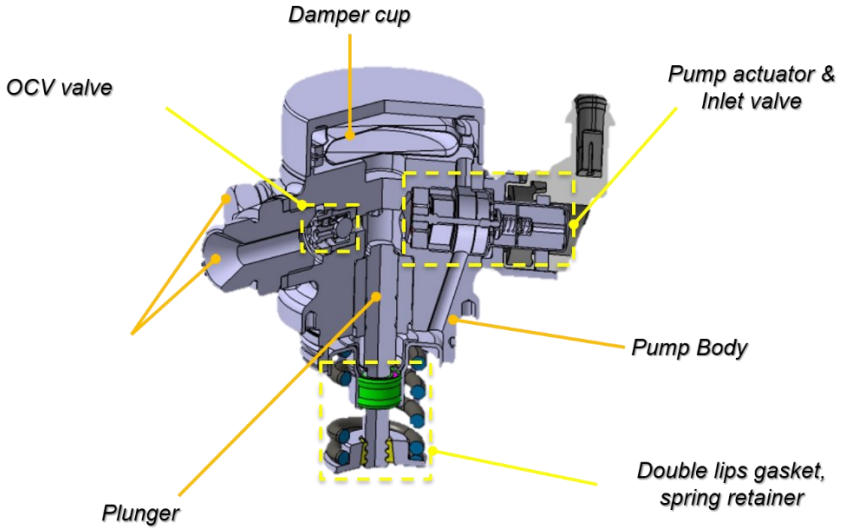


Figura 28 schema pompa

Grazie alle dimensioni compatte, al peso ridotto e alla flessibilità di personalizzazione dell'interfaccia (flangia di fissaggio, ingresso e uscita carburante e collegamento elettrico), può essere facilmente installato su diverse tipologie di motori, soddisfacendo le esigenze di layout sempre più piccoli.

Essendo in grado di generare pressioni da 40 a 200 bar, questa pompa può soddisfare le esigenze dei sistemi attualmente in fase di sviluppo e produzione, offrendo una vasta gamma di pressioni; incorpora una valvola di massima per la protezione del circuito di alta pressione e uno smorzatore di ingresso che limita le fluttuazioni di pressione sul circuito di alimentazione.

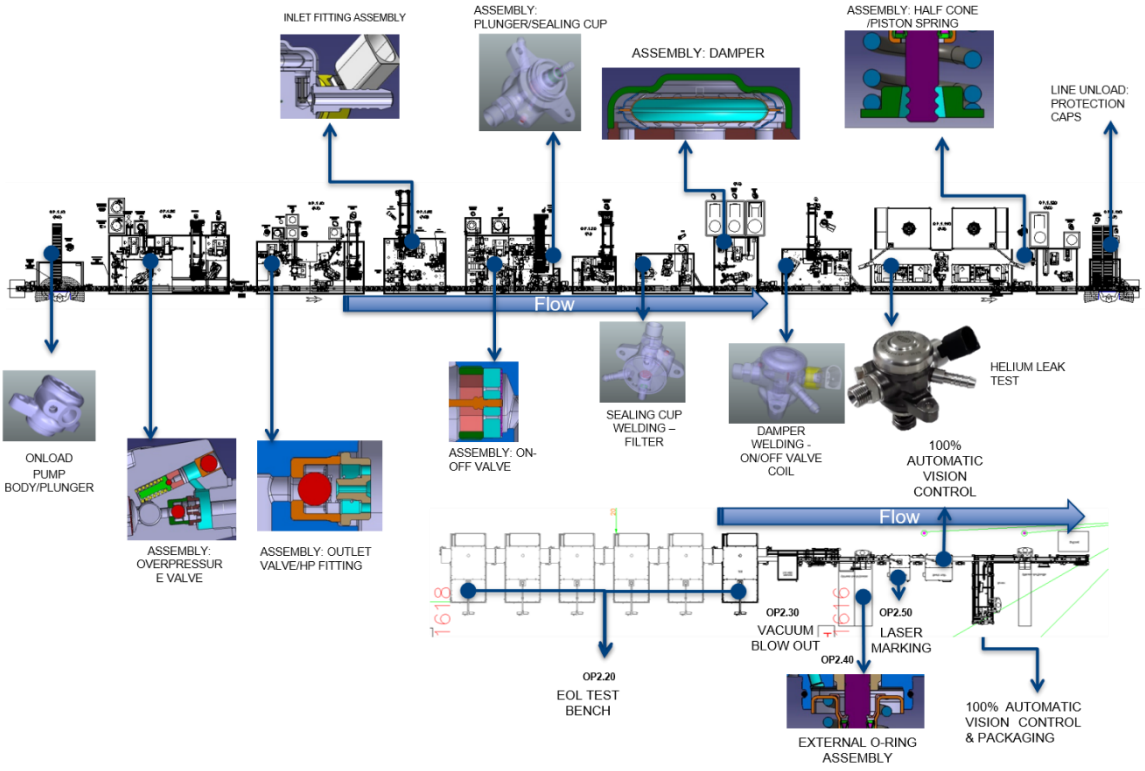


Figura 29 schema linea di assemblaggio

Le operazioni da effettuare sono, nell'ordine:

- caricamento della struttura e del pistone
- assemblaggio della Pressure Relief Valve
- assemblaggio della valvola di uscita e del raccordo ad alta pressione
- montaggio del raccordo di aspirazione
- assemblaggio della valvola on/off
- montaggio di pistone, coppa di tenuta e filtro
- montaggio del regolatore
- assemblaggio del mezzo cono e della molla e del fermo del pistone
- EOL test
- Assemblaggio O-ring.

Linguaggio di programmazione Python

Con più di un milione di siti web clienti, Python rappresenta il linguaggio spesso preferito dagli sviluppatori e dai data scientist che hanno bisogno di applicare tecniche statistiche o analisi dei dati nel loro lavoro.

Figura 30 simbolo Python

La combinazione di sintassi coerente, tempi di sviluppo brevi e flessibilità lo rendono adatto allo sviluppo di modelli di previsione sofisticati che possono essere collegati direttamente ai sistemi di produzione e all'apprendimento automatico. Dopotutto Python è un linguaggio di programmazione, orientato agli oggetti e di alto livello con semantica dinamica: queste caratteristiche gli permettono di esprimere idee molto potenti in pochissime righe di codice pur essendo molto leggibili. Questo è possibile anche grazie all'ampio set di librerie, ossia insiemi di routine e funzioni scritte che svolgono un determinato compito, che possiede e può richiamare a seconda delle necessità. Le librerie vengono spesso confuse con i termini framework e packages. Prima di vedere quali sono le più diffuse per il machine learning, bisogna distinguere i termini tenendo conto di quanto stabilito dalla documentazione presente sul sito di Python.

Differenza tra librerie, packages e framework

Innanzitutto, si può considerare la libreria come un insieme di moduli. Ogni modulo contiene istruzioni e definizioni semplici. L'accorpamento di vari moduli, quindi di istruzione codice, costituisce una libreria. Spesso i moduli sono già stati scritti da altri sviluppatori, e non c'è bisogno di ripartire da capo ogni volta. Il loro scopo è quello di semplificare le attività, aiutando gli sviluppatori a scrivere solo poche righe anziché una grande quantità di comandi. Il codice delle librerie richiama classi e metodi che normalmente definiscono operazioni specifiche in un'area del dominio. Ad esempio, ci sono alcune librerie di matematica che possono far sì che lo sviluppatore chiami semplicemente la funzione senza ripetere l'implementazione di come funziona un algoritmo.

Per capire cosa sono i packages si deve pensare alle cartelle (conosciute col nome directory), dove vengono salvati i file nel computer. Di solito non tutti i file vengono archiviati nella stessa posizione. Si utilizza una gerarchia di directory ben organizzata per un accesso più semplice. File simili sono tenuti nella stessa directory, ad esempio, potremmo conservare tutti i brani musicali nella directory "musica". Analogamente a questo, Python ha packages per directory e moduli per i file. Dato che una directory può contenere sottodirectory e file, similmente, un pacchetto Python può avere sotto-pacchetti e moduli. Una directory deve contenere un file chiamato __init__.py in modo che Python lo consideri come un pacchetto. Questo file può essere lasciato vuoto ma generalmente il codice di inizializzazione per quel pacchetto viene inserito in questo file.

A differenza delle librerie, per framework si intende "un'astrazione", in cui il software che fornisce funzionalità generiche può essere modificato selettivamente da un ulteriore codice scritto dall'utente, fornendo così un software specifico per l'applicazione. Si può considerare il framework come uno strumento software che fornisce un modo per creare ed eseguire applicazioni web e per farlo si avvale spesso di librerie e packages. Utilizzando un framework web non è necessario scrivere codice per conto proprio e perdere tempo cercando possibili errori di calcolo e bug. All'inizio dello sviluppo web, tutte le applicazioni erano codificate a mano e solo lo sviluppatore di una determinata app poteva cambiarlo o distribuirlo. I framework Web hanno introdotto un modo semplice per uscire da questa trappola. Dal 1995, tutte le seccature legate al cambiamento della struttura di un'applicazione sono state messe in ordine a causa della comparsa di una prestazione

generale; successivamente a questo sono apparse le lingue specifiche del web. La loro varietà ora funziona bene sia per pagine web statiche che dinamiche.

Possiamo avere tre tipologie di Framework web:

- Server-side: definito anche come framework backend, sono applicazioni software che facilitano la scrittura, la manutenzione e la scalabilità delle applicazioni web. Forniscono strumenti e librerie che semplificano le comuni attività di sviluppo Web, inclusi gli URL di routing ai gestori appropriati, l'interazione con i database, le sessioni di supporto e l'autorizzazione dell'utente, l'output di formattazione (ad esempio HTML, JSON, XML) e il miglioramento della sicurezza dagli attacchi web.
- Client-side: definito anche framework frontend, consiste in un pacchetto costituito da una struttura di file e cartelle di codice standard (HTML, CSS, documenti JS ecc.). Si occupa essenzialmente delle parti rivolte verso l'esterno di un sito Web o di un'applicazione web. In breve, ciò che un utente vede quando apre l'app.
- Full-stack Framework: è una combinazione di entrambe le estremità frontend e backend. Uno sviluppatore full-stack è un tuttofare, ed è responsabile di tutti i livelli di sviluppo, da come il server è impostato per il CSS relativo alla progettazione. C'è da dire che è complesso gestire entrambe le parti.

Librerie più popolari nel machine learning

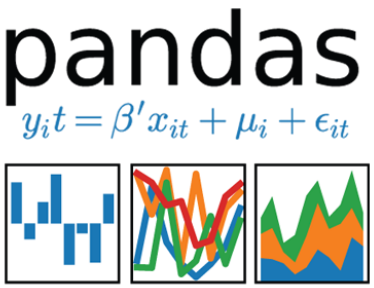
Vediamo ora quali che sono le librerie più importanti di Python utilizzate nel machine learning.

Scikit-learn

Fornisce una gamma di algoritmi di apprendimento supervisionato e non supervisionato tramite un'interfaccia coerente in Python. È così possibile risolvere facilmente algoritmi di regressione, di classificazione e clustering. Anche attività quali la trasformazione di dati, la selezione delle funzionalità e i metodi di ensemble possono essere implementate in poche righe.

Pandas

Libreria che fornisce gli strumenti per l'analisi dei dati nel linguaggio Python. Il pacchetto è open source e viene fornito con diverse strutture dati che possono essere utilizzate per diverse attività di manipolazione dei dati. Pandas è una libreria molto popolare per il recupero e la preparazione dei dati per l'uso futuro in altre librerie per il machine learning come Scikit-learn o Tensorflow. Permette inoltre di recuperare facilmente i dati da diverse fonti: database SQL, testo, CSV, Excel, file JSON e molti altri formati meno popolari. Una volta che i dati sono in memoria, ci sono dozzine di operazioni diverse per analizzare, trasformare, recuperare i valori mancanti, pulire il set di dati, nonché operazioni tipo SQL e un set di funzioni statistiche per eseguire anche una semplice analisi.



NumPy

sta per Numeric Python e rappresenta il pacchetto fondamentale per il calcolo scientifico con Python. Questa è ovviamente una delle più grandi librerie di calcolo matematico e scientifico per Python. Una delle funzionalità più importanti di NumPy è la sua interfaccia Array. Questa interfaccia può essere utilizzata per esprimere immagini, onde sonore o altri flussi binari grezzi come matrici di numeri reali con dimensione N. La conoscenza di NumPy è molto importante per l'apprendimento automatico e la scienza dei dati.

Matplotlib

L'apprendimento automatico migliore e più sofisticato è privo di significato se non puoi comunicarlo ad altre persone. Quindi, come si fa a trasformare effettivamente il valore da tutti questi dati che si hanno? È qui che Matplotlib viene in soccorso. È una libreria Python standard utilizzata per la creazione di diagrammi e grafici 3D. È piuttosto di basso livello, il che significa che richiede più comandi per generare grafici e figure piacevoli rispetto ad alcune librerie avanzate. Tuttavia, il rovescio della medaglia è la flessibilità. Con qualche comando si può creare praticamente qualsiasi tipo di grafico desiderato con Matplotlib. È possibile crearne di diversi tipi, dagli istogrammi ai grafici a dispersione, fino ai grafici con coordinate non cartesiane.

Theano

Rappresenta una libreria Python che consente di valutare, ottimizzare e definire espressioni matematiche che coinvolgono efficacemente gli array multidimensionali (è simile a NumPy). Questa libreria ottimizza l'utilizzo della CPU e della GPU e migliora le prestazioni del calcolo intensivo dei dati, in quanto il codice Theano è scritto in modo tale da sfruttare le funzioni utilizzate da un compilatore del computer. È una delle librerie di deep learning più utilizzate fino ad oggi, anche se l'ultima versione è stata rilasciata nel 2017.

TensorFlow

Secondo diversi studi Tensorflow è un'altra delle librerie più utilizzate da chi si affaccia nel mondo del deep learning. Tale libreria è stata sviluppato da Google, e quasi tutte le sue applicazioni utilizzano Tensorflow per l'apprendimento automatico. Se stai utilizzando le foto di Google o la ricerca vocale di Google, indirettamente stai utilizzando i modelli creati utilizzando Tensorflow. Questo è un framework computazionale per esprimere algoritmi che coinvolgono un gran numero di operazioni di tensori, poiché le reti neurali possono essere espresse come grafici computazionali tramite una serie di operazioni sui tensori. I tensori sono matrici N-dimensionali che rappresentano i nostri dati.

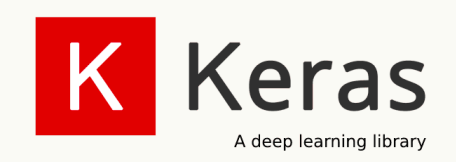
Pytorch

Rappresenta una popolare libreria di Deep Learning creata da Facebook. Oltre alla CPU, supporta calcoli accelerati dalla GPU. La libreria è focalizzata sul portare agli utenti un'esperienza di modellazione veloce e flessibile e ha ottenuto molto seguito nella comunità di Deep Learning. Rispetto a Tensorflow, è più facile da imparare ad usare. Il rovescio della medaglia è che PyTorch è meno maturo di Tensorflow, ma la comunità sta crescendo rapidamente e ci sono già molti materiali didattici e tutorial.

Keras

È una delle librerie di apprendimento automatico più interessanti. Gli esperti consigliano di partire con questa libreria, a chi inizia a studiare il machine learning, poiché fornisce un modo più semplice per esprimere reti neurali. Fornisce inoltre alcune utilità per l'elaborazione di set di dati, la compilazione di modelli, la valutazione dei risultati, la visualizzazione di grafici e molto altro. Gli attuali vantaggi di Keras, a differenza di PyTorch, sono che è più maturo, ha una community più grande e un sacco di tutorial pronti all'uso. Inoltre, può utilizzare Tensorflow. D'altra parte, PyTorch fornisce alcune

caratteristiche molto interessanti (come il debug interattivo o la definizione dinamica del grafico) e sta crescendo rapidamente.



Le librerie Python appena viste semplificano di molto il machine learning (ce ne sono anche altre non citate molto utili), e hanno permesso di migliorare i risultati ottenuti dall'intelligenza artificiale.

Il linguaggio

Python è un esempio di linguaggio di alto livello; altri linguaggi di alto livello sono il C, il C++, il Perl ed il Java.

Esistono anche linguaggi di basso livello, talvolta chiamati "linguaggi macchina" o "linguaggi assembly". In modo non del tutto corretto si può affermare che i computer possono eseguire soltanto programmi scritti in linguaggi di basso livello, quelli scritti in un linguaggio di alto livello devono essere elaborati prima di poter essere eseguiti.

Questo processo di elaborazione impiega del tempo e rappresenta un piccolo svantaggio dei linguaggi di alto livello, ma i vantaggi sono d'altra parte enormi. In primo luogo, è molto più facile programmare in un linguaggio ad alto livello, questi tipi di programmi sono più veloci da scrivere, più corti e facilmente leggibili, ed è più probabile che siano corretti. In secondo luogo, i linguaggi di alto livello sono portabili: portabilità significa che essi possono essere eseguiti su tipi di computer diversi con poche o addirittura nessuna modifica.

I programmi scritti in linguaggi di basso livello possono essere eseguiti solo su un tipo di computer e devono essere riscritti per essere trasportati su un altro sistema. Questi vantaggi sono così evidenti che quasi tutti i programmi sono scritti in linguaggi di alto livello, lasciando spazio ai linguaggi di basso livello solo in poche applicazioni specializzate.

I programmi di alto livello vengono trasformati in programmi di basso livello eseguibili dal computer tramite due tipi di elaborazione: l'interpretazione e la compilazione. Un interprete legge il programma di alto livello e lo esegue, trasformando ogni riga di istruzioni in un'azione. L'interprete elabora il programma un po' alla volta, alternando la lettura delle istruzioni all'esecuzione dei comandi che le istruzioni descrivono. Un compilatore legge il programma di alto livello e lo traduce completamente in basso livello, prima che il programma possa essere eseguito. In questo caso il programma di alto livello viene chiamato codice sorgente, e la sua traduzione codice oggetto o eseguibile. Dopo che un programma è stato compilato può essere eseguito ripetutamente senza che si rendano necessarie ulteriori compilazioni finché non ne viene modificato il codice.

Python è considerato un linguaggio interpretato perché i programmi Python sono eseguiti da un interprete. Ci sono due modi di usare l'interprete: a linea di comando o in modo script. Nel primo caso si scrivono i programmi una riga alla volta: dopo avere scritto una riga di codice alla pressione di Invio l'interprete la analizza subito ed elabora immediatamente il risultato, eventualmente stampandolo a video.

```
>>> print 1 + 1
2
```

La prima linea di questo esempio inizia con >>>: questa è l'indicazione (chiamata "prompt") che l'interprete usa per indicare la sua disponibilità ad accettare comandi. Inserendo "print 1 + 1" l'interprete risponde con "2". In alternativa alla riga di comando si può scrivere un programma in un file (detto script) ed usare l'interprete per eseguire il contenuto del file.

Cos'è un programma

Un programma è una sequenza di istruzioni che specificano come effettuare un'elaborazione. Essa può essere sia di tipo matematico (per esempio la soluzione di un sistema di equazioni o il calcolo delle radici di un polinomio) che simbolico (per esempio la ricerca e sostituzione di un testo in un documento). I dettagli sono diversi per ciascun linguaggio di programmazione, ma un piccolo gruppo di istruzioni è praticamente comune a tutti:

- input: ricezione di dati da tastiera, da file o da altro dispositivo;
- output: scrittura di dati su video, su file o trasmissione ad altro dispositivo;
- matematiche: esecuzione di semplici operazioni matematiche, quali l'addizione e la sottrazione;
- condizionali: controllo di alcune condizioni ed esecuzione della sequenza di istruzioni appropriata;
- ripetizione: ripetizione di un'azione, di solito con qualche variazione.

Cos'è il debug

La programmazione è un processo complesso e dato che esso è fatto da esseri umani spesso comporta errori. Gli errori di programmazione sono chiamati bug ed il processo della loro ricerca e correzione è chiamato debug. Sono tre i tipi di errore nei quali si incorre durante la programmazione: gli errori di sintassi, gli errori in esecuzione e gli errori di semantica. È utile distinguerli per poterli individuare più velocemente.

Errori di sintassi: Python può eseguire un programma solo se il programma è sintatticamente corretto, altrimenti l'elaborazione fallisce e all'interprete ritorna un messaggio di errore. La sintassi si riferisce alla struttura di un programma e alle regole concernenti la sua struttura. In italiano, per fare un esempio, una frase deve iniziare con una lettera maiuscola e terminare con un punto, altrimenti si commette un errore di sintassi.

Python non è per niente permissivo: se c'è un singolo errore di sintassi da qualche parte nel programma, stamperà un messaggio d'errore e ne interromperà l'esecuzione, rendendo impossibile proseguire.

Errori in esecuzione: Il secondo tipo di errore è quello in esecuzione (o "runtime"), così chiamato perché l'errore non appare finché il programma non è eseguito. Questi errori sono anche chiamati eccezioni perché indicano che è accaduto qualcosa di eccezionale nel corso dell'esecuzione (per esempio si è cercato di dividere un numero per zero).

Errori di semantica: Il terzo tipo è l'errore di semantica. Se c'è un errore di questo tipo il programma verrà eseguito senza problemi, nel senso che il computer non genererà messaggi d'errore durante l'esecuzione, ma il risultato non sarà ciò che ci si aspettava, sarà qualcosa di diverso, e questo qualcosa è esattamente ciò che è stato detto di fare al computer. Il problema sta nel fatto che il programma che è stato scritto non è quello che si desiderava scrivere: il significato del programma (la sua semantica) è sbagliato. L'identificazione degli errori di semantica è un processo complesso perché richiede di lavorare in modo inconsueto, guardando i risultati dell'esecuzione e cercando di capire cosa il programma ha fatto di sbagliato per ottenerli.

Linguaggi formali e naturali

I linguaggi naturali sono le lingue parlate, tipo l'inglese, l'italiano, lo spagnolo. Non sono stati "progettati" da qualcuno e anche se è stato imposto un certo ordine nel loro sviluppo si sono evoluti naturalmente. I linguaggi formali sono linguaggi progettati per specifiche applicazioni. Per fare qualche esempio, la notazione matematica è un linguaggio formale particolarmente indicato ad esprimere relazioni tra numeri e simboli; i chimici usano un linguaggio formale per rappresentare la struttura delle molecole; i linguaggi di programmazione sono linguaggi formali che sono stati progettati per esprimere elaborazioni. Questi tendono ad essere piuttosto rigidi per quanto riguarda la sintassi: 3 + 3 = 6 è una dichiarazione matematica sintatticamente corretta, mentre 3 = ÷6$ non lo è, H2O è un simbolo chimico sintatticamente corretto contrariamente a 2Zz. Le regole sintattiche si possono dividere in due categorie: la prima riguarda i token, la seconda la struttura. I token sono gli elementi di base del linguaggio (quali possono essere le parole in letteratura, i numeri in matematica e gli elementi chimici in chimica). Uno dei problemi con 3 = ÷6$ è che $ non è un token valido in matematica; 2Zz non è valido perché nessun elemento chimico è identificato dal simbolo Zz. Il secondo tipo di regola riguarda la struttura della dichiarazione, cioè il modo in cui i token sono disposti. La dichiarazione 3

= ÷6 è strutturalmente non valida perché un segno ÷ non può essere posto immediatamente dopo un segno =. Allo stesso modo l'indice nelle formule chimiche deve essere indicato dopo il simbolo dell'elementi chimico, non prima, e quindi l'espressione 2Zz non è valida.

Valori e tipi

Un valore è una delle cose fondamentali manipolate da un programmatore, come lo sono una lettera dell'alfabeto nella scrittura o un numero in matematica. I valori che vediamo sono "Hello World!" e "2", quest'ultimo il risultato ottenuto quando abbiamo sommato 1+1. Questi valori appartengono a tipi diversi: "2" è un intero, e "Hello, World!"è una stringa, così chiamata perché contiene una serie (o "stringa") di caratteri. L'interprete può identificare le stringhe perché sono racchiuse tra virgolette. L'istruzione "print" funziona sia per le stringhe che per gli interi. Se non si è sicuri del tipo di un valore, l'interprete può dircelo:

```
>>> type("Hello, World!")
<type 'string'>

>>> type(17)
<type 'int'>
```

Ovviamente le stringhe appartengono al tipo string e gli interi al tipo int. Non è invece intuitivo il fatto che i numeri con il punto decimale appartengano al tipo float: questi numeri sono rappresentati in un formato chiamato virgola mobile o floating-point.

```
>>> type(3.2)
<type 'float'>
```

Cosa dire di numeri come "17" e "3.2"?
Sembrano effettivamente dei numeri, ma sono racchiusi tra virgolette e questo sicuramente significa qualcosa, infatti, non siamo in presenza di numeri ma di stringhe:

```
>>> type("17")
<type 'string'>

>>> type("3.2")
<type 'string'>
```

Variabili

Una delle caratteristiche più potenti in un linguaggio di programmazione è la capacità di manipolare variabili. Una variabile è un nome che si riferisce ad un valore. L'istruzione di assegnazione crea nuove variabili e assegna loro un valore:

```
>>> messaggio = "Come va?"
>>> n = 17
>>> pi = 3.14159
```

Questo esempio effettua tre assegnazioni. La prima assegna la stringa "Come va?" ad una nuova variabile chiamata messaggio. La seconda assegna l'intero 17 alla variabile "n" e la terza assegna il valore 3,14159 alla variabile "pi". L'istruzione print funziona anche con le variabili:

```
>>> print messaggio
Come va?
>>> print n
17
>>> print pi
3.14159
```

ed in ogni caso il risultato è il valore della variabile. Anche le variabili hanno il tipo; ancora una volta possiamo chiedere all'interprete a quale tipo ogni variabile appartenga:

```
>>> type(message)
<type 'string'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Il tipo di una variabile è il tipo di valore cui essa si riferisce.

Risultati

Risultati OEE

La prima cosa da fare è installare le varie librerie necessarie a far funzionare il modello, in questo caso xlrd (per l'importazione del dataset dal file Excel), sklearn, pandas e numpy.
Si procede poi con l'importazione e la modifica del dataset, suddividendo inoltre i dati in due parti: una dedicata all'allenamento e una alla fase di test. La variabile X corrisponde agli input, la y agli output. Questa funzione crea quattro variabili, gli input e gli output associati al train (X_train, y_train) e quelli associati al test (X_test, y_test).

```
model = pd.read_excel("venv/DATASET.xlsx")
model = model.drop(['Pezzi totali prodotti', 'TOTALE PEZZI BUONI', 'FTQ'], axis=1)
y = model['OEE']
X = model.drop('OEE', axis=1)
feature_list = list(model.columns)
print(feature_list)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print(y_train.describe)
```

Ho deciso di utilizzare il 30% dei dati per la parte di test, viste le ridotte dimensioni del dataset.

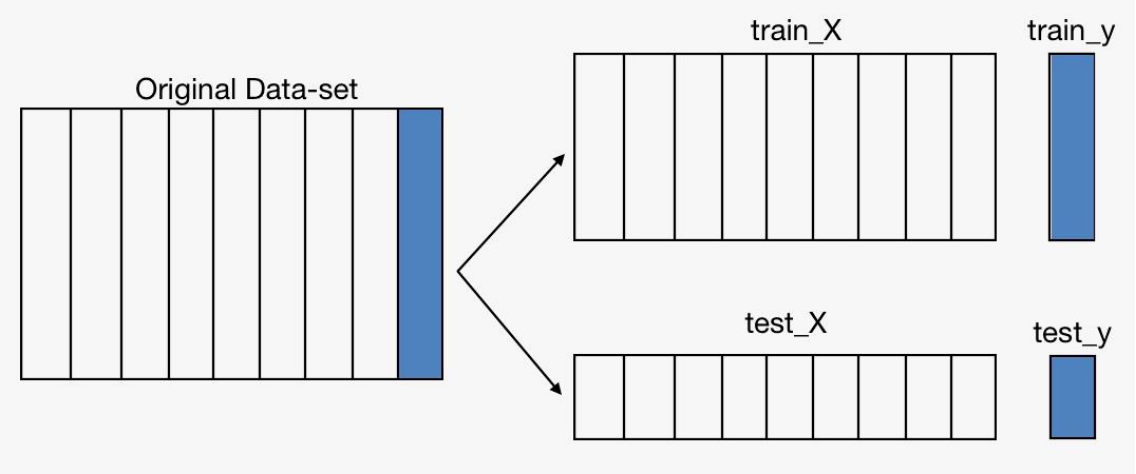


Figura 31 split del dataset

Il comando drop serve ad eliminare una riga (con axis=0) o una colonna (con axis=1) del dataset, in questo caso è stata usata per eliminare le colonne contenenti le feature da non utilizzare nell'addestramento.
Dopo aver preparato il dataset, una delle tecniche di Scikit-Learn più utilizzate per valutare un modello decisionale durante questa fase di allenamento è la cosiddetta K-fold cross-

validation. Questa suddivide in modo casuale il training set in K sottoinsiemi, chiamati folds (pieghe), e successivamente allena e valuta l'algoritmo in K iterazioni, dedicando K-1 folds per l'allenamento e 1 fold per la validazione, sempre diversi ad ogni iterazione. In questo caso ho adottato un valore di K pari a 10, il che vuol dire avere il training set suddiviso in dieci folds e dieci iterazioni.

In questo modo, oltre ad abbassare di molto il rischio di overfitting, si andrà ad evitare anche il problema del campionamento asimmetrico, in quanto, in una o nell'altra iterazione, si utilizzeranno tutti i dati per addestrare il modello.

Figura 32 k-fold validation

Si procede poi con ottimizzazione degli iperparametri, processo che prende il nome di tuning. Per fare ciò esistono due funzioni di scikit-learn: RandomizedSearchCV e GridSearchCV. Entrambe svolgono questo compito automaticamente, ma con modalità diverse.

La GridSearchCV, assegnati dei valori, o un range di valori, agli iperparametri prova ogni combinazione possibile, trovando la migliore. Questo procedimento, pur essendo molto efficace, richiede molto tempo e molto spreco di risorse. Per velocizzare il processo di hyperparameter tuning si può quindi utilizzare, in un primo momento, la RandomizedSearchCV per restringere il range di parametri. Questa funzione prova solamente un numero di combinazioni, scelto dal programmatore, con valori scelti causalmente (random) tra tutti quelli assegnati. In questo modo, come si può facilmente

intuire, si riduce di molto il tempo di computazione, andando ad eseguire solamente decine di iterazioni al posto di migliaia.
Sia la RandomizedSearchCV che la GridSearchCV, come lascia intuire il nome, utilizzano la Cross Validation (CV) internamente, allenando sul training set e validando sul validation set il modello.
Per eseguire correttamente il procedimento bisogna quindi definire il range di valori degli iperparametri da ottimizzare, definendo adeguatamente delle funzioni, come mostrato in figura.

```
n_estimators = [int(x) for x in np.linspace(start=10, stop=1000, num=20)]
max_features = ["auto", "sqrt", "log2"]
max_depth = [int(x) for x in np.linspace(10, 1000, num=20)]
min_samples_split = [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 50, 100]
min_samples_leaf = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 50, 100]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

n_estimator rappresenta il numero di predittori (Decision Trees) che l'algoritmo costruisce prima di fare le operazioni di votazioni e predizioni e, in generale, un più alto numero di alberi incrementa le performance e permette delle predizioni più stabili a scapito di un maggior tempo computazionale.
max_features è il massimo numero di features che Random Forest considera per suddividere un nodo.
max_depth rappresenta la profondità di ogni albero della foresta definita in livelli e, in generale, più profondo è un Decision Tree e maggiori split possiede e maggiori informazioni riesce a catturare.
min_samples_split rappresenta il numero minimo di istanze richieste per suddividere un nodo interno e può variare considerando almeno un dato per nodo fino a prendere tutte le istanze in ogni nodo; incrementando tale parametro, ogni albero della foresta diventa più vincolato.
min_sample_leaf rappresenta il numero minimo di istanze richieste nei nodi foglia, quest'ultimi costituenti la base dell'albero.

Successivamente si passa alla definizione del tipo di struttura alla base del modello, nel caso in esame RandomForestRegressor, e all'impostazione dei parametri e delle funzioni per l'addestramento.

```
rf = RandomForestRegressor(random_state=0)
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid, n_iter=30,
                               cv=10, verbose=2, n_jobs=-1)
rf_random.fit(X_train, y_train)
print(rf_random.best_params_)
random_best_train = rf_random.best_estimator_.predict(X_train)
random_best_test = rf_random.best_estimator_.predict(X_test)
```

n_jobs indica alla macchina quanti processori (core) le è permesso usare ed è sempre impostato al valore -1, che significa nessun limite di utilizzo dei core.
random_state è utilizzato per rendere replicabile l'uscita del modello, il quale produrrà sempre gli stessi risultati per un certo valore di questo parametro e se lavora con gli stessi iperparametri e training data.
La funzione fit dà in pasto all'algoritmo i dati di addestramento, mentre la funzione best_params_, come argomento dalla funzione print, consente di visualizzare la miglior combinazione di iperparametri.
Per controllare la qualità delle predizioni effettuate utilizzando una certa combinazione di parametri è possibile usare le funzioni r2_score, mean_squared_error e mean_absolute_error. sono stati analizzati sia il train set che il test set, in modo da accorgersi della presenza di overfitting o underfitting.
r2_score: coefficiente di determinazione, è una proporzione tra la variabilità dei dati e la correttezza del modello statistico utilizzato. Esso misura la frazione della varianza della variabile dipendente espressa dalla regressione. Non esiste una definizione concordata di r2, ad esempio nelle regressioni lineari semplici esso è semplicemente il quadrato del coefficiente di correlazione. r2 varia tra 0 ed 1: quando è 0 il modello utilizzato non spiega per nulla i dati; quando è 1 il modello spiega perfettamente i dati.

```
r2_score_train = r2_score(y_train, random_best_train)
print('r2_train:', round(r2_score_train, 3))
r2_score_test = r2_score(y_test, random_best_test)
print('r2_test:', round(r2_score_test, 3))
```

mean_squared_error: errore quadratico medio, ci dà una misura per giudicare la qualità di uno stimatore in termini della sua variazione e della sua distorsione. È uguale alla somma

della varianza e del quadrato del bias di uno stimatore. Più il valore è piccolo, migliore sarà la corrispondenza tra il valore predetto e il valore reale.

```
mse_train = mean_squared_error(y_train, random_best_train)
rmse_train = np.sqrt(mse_train)
print('RMSE_train:', round(rmse_train, 3))
mse_test = mean_squared_error(y_test, random_best_test)
rmse_test = np.sqrt(mse_test)
print('RMSE_test:', round(rmse_test, 3))
```

mean_absolute_error: errore assoluto medio, esprime la differenza tra il valore predetto e il valore reale. Per un modello ottimale questo valore deve essere il minore possibile.

```
mae_train = mean_absolute_error(y_train, random_best_train)
print('MAE_train:', round(mae_train, 3))
mae_test = mean_absolute_error(y_test, random_best_test)
print('MAE_test:', round(mae_test, 3))
```

I risultati ottenuti dopo una prima analisi sono riportati di seguito

'n_estimators': 947, 'min_samples_split': 8, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 426			
	r2	MAE	RMSE
TRAIN	0.98	0.015	0.026
TEST	0.943	0.025	0.044

'n_estimators': 114, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 583			
	r2	MAE	RMSE
TRAIN	0.976	0.016	0.029
TEST	0.942	0.025	0.045

'n_estimators': 635, 'min_samples_split': 6, 'min_samples_leaf': 6, 'max_features': 'auto', 'max_depth': 374			
	r2	MAE	RMSE
TRAIN	0.951	0.023	0.041
TEST	0.932	0.028	0.049

Riducendo il range degli iperparametri, dopo una rapida analisi dei risultati sopra riportati, si ha un primo miglioramento nel test set. Ciò significa che si è riuscito a ridurre l'overfitting attraverso una ulteriore ottimizzazione dei parametri.

'n_estimators': 116, 'min_samples_split': 7, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 560			
	r2	MAE	RMSE
TRAIN	0.98	0.014	0.026
TEST	0.943	0.025	0.044

'n_estimators': 269, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 560			
	r2	MAE	RMSE
TRAIN	0.982	0.013	0.025
TEST	0.942	0.026	0.045

'n_estimators': 116, 'min_samples_split': 8, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 494			
	r2	MAE	RMSE
TRAIN	0.979	0.015	0.027
TEST	0.944	0.025	0.044

Avendo dei range di valori inferiori, si può ora passare alla Grid Search, di seguito le stringhe di codice usate.

```
n_estimators = [int(x) for x in np.linspace(start=115, stop=270, num=50)]
max_features = ["auto"]
max_depth = [560]
min_samples_split = [6, 7, 8]
min_samples_leaf = [1]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

rf = RandomForestRegressor()
rf_grid = GridSearchCV(estimator=rf, param_grid=random_grid, cv=10, verbose=2, n_jobs=-1)
rf_grid.fit(X_train, y_train)
print(rf_grid.best_params_)
```

Ed i risultati ottenuti dalla Grid Search

'max_depth': 560, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 7, 'n_estimators': 124			
	r2	MAE	RMSE
TRAIN	0.981	0.014	0.026
TEST	0.944	0.025	0.044

'max_depth': 560, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 6, 'n_estimators': 121			
	r2	MAE	RMSE
TRAIN	0.981	0.014	0.025
TEST	0.943	0.026	0.044

'max_depth': 560, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 6, 'n_estimators': 213			
	r2	MAE	RMSE
TRAIN	0.983	0.013	0.024
TEST	0.943	0.025	0.044

Restringendo ulteriormente il campo si giunge infine alla migliore combinazione trovata:

'max_depth': 560, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 6, 'n_estimators': 153			
	r2	MAE	RMSE
TRAIN	0.982	0.013	0.025
TEST	0.944	0.025	0.044

Si è riuscita a raggiungere un'accuratezza sui dati di test del 94,4% con un errore medio di 0,025 in un range tra 0 e 1, con una varianza dei risultati molto bassa.

A questo punto si può andare ad analizzare da quali features dipende maggiormente l'output, in modo da eliminare quelle che hanno un'importanza percentuale bassa, in modo da velocizzare il tempo di computazione. Per fare ciò si utilizza una funzione di scikit learn, chiamata feature_importances_ che dà in output, per ogni feature, la sua importanza nell'ottenimento della previsione espressa con un valore da 0 a 1. Le stringhe di codice sono le seguenti.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
rfr = RandomForestRegressor(n_estimators=153, min_samples_split=6, min_samples_leaf=1,
                            max_depth=560, random_state=0)
rfr.fit(X_train, y_train)

importances = list(rfr.feature_importances_)
feature_importances = [(feature, round(importance, 3))
                       for feature, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key=lambda x: x[1], reverse=True)
[print('{}:{}'.format(*pair)) for pair in feature_importances]
```

Seguono i risultati della feature_importances_. Ho riportato solo le variabili con un'importanza superiore allo 0,1%.

```
TEMPO OPERATIVO:0.941
Mese:0.013
Giorno:0.009
GUASTI - RISOLTI PM:0.005
Disponibilità attuale (MINUTI TURNO):0.004
MENSA:0.004
MICROFERMATE:0.004
MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE:0.003
Fermo problemi Qualità:0.003
Totale Min riparazione PM:0.003
Turno:0.001
MANUTENZIONE PM PROGRAMMATA:0.001
SET-UP Cambio Tipo:0.001
GUASTI - RISOLTI AM:0.001
Totale Min riparazione AM:0.001
OP_OP_40.1 PM:0.001
OP_OP_80 PM:0.001
OP_OP_120 PM:0.001
```

Si può andare ora a costruire un modello utilizzando in input solo le features con un'importanza superiore allo 0,5%, in modo da ridurre significativamente il numero di features, da 48 a 4, riducendo al minimo il tempo di computazione.

```
model = pd.read_excel("venv/DATASET.xlsx")
model1 = ['TEMPO OPERATIVO', 'Mese', 'Giorno', 'GUASTI - RISOLTI PM', 'OEE']
model = model[model1]
X = model.drop('OEE', axis=1).values

y = model['OEE'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0)

rf = RandomForestRegressor(n_estimators=153, min_samples_split=6,
                           min_samples_leaf=1, max_depth=560, random_state=0)
rf.fit(X_train, y_train)
rf_train = rf.predict(X_train)
rf_test = rf.predict(X_test)

print('r2_train:', round(r2_score(y_train, rf_train), 3))
mse_train = mean_squared_error(y_train, rf_train)
rmse_train = np.sqrt(mse_train)
print('RMSE_train:', round(rmse_train, 3))
mae_train = mean_absolute_error(y_train, rf_train)
print('MAE_train:', round(mae_train, 3))

print('r2_test:', round(r2_score(y_test, rf_test), 3))
mse_test = mean_squared_error(y_test, rf_test)
rmse_test = np.sqrt(mse_test)
print('RMSE_test:', round(rmse_test, 3))
mae_test = mean_absolute_error(y_test, rf_test)
print('MAE_test:', round(mae_test, 3))
```

Di seguito i risultati ottenuti dal modello con le sole feature importanti e quelli usando per l'addestramento tutte le variabili.

SOLO VARIABILI SOPRA 0,5%			
	r2	MAE	RMSE
TRAIN	0.976	0.016	0.029
TEST	0.917	0.03	0.053

TUTTE LE VARIABILI			
	r2	MAE	RMSE
TRAIN	0.982	0.013	0.025
TEST	0.944	0.025	0.044

Si può notare che le differenze sono molto piccole, nell'ordine dell'1-2%. Perciò se il parametro più importante è il tempo, si possono ridurre le variabili senza ottenere un significativo peggioramento dei risultati.
Successivamente ho ripetuto lo stesso procedimento, creando modelli anche per il numero totale di pezzi prodotti e per il numero totale di pezzi buoni.

Risultati pezzi totali prodotti

'max_depth': 210, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 126			
	r2	MAE	RMSE
TRAIN	0.991	11.276	27.129
TEST	0.979	21.689	41.003

Utilizzando solo le variabili importanti (sopra allo 0,5%) che sono:

TEMPO OPERATIVO:0.969

MANUTENZIONE PM PROGRAMMATA:0.007

Mese:0.005

MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE:0.005

'max_depth': 210, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 126			
	r2	MAE	RMSE
TRAIN	0.986	16.677	34.116
TEST	0.974	25.135	45.707

Risultati totale pezzi buoni

'max_depth': 38, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 9, 'n_estimators': 318			
	r2	MAE	RMSE
TRAIN	0.98	26.807	49.441
TEST	0.964	39.073	65.439

Utilizzando solo le variabili importanti (sopra allo 0,5%) che sono:

TEMPO OPERATIVO:0.965

Mese:0.009

Giorno:0.006

MICROFERMATE:0.006

MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE:0.005

'max_depth': 38, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 9, 'n_estimators': 318			
	r2	MAE	RMSE
TRAIN	0.978	28.796	52.221
TEST	0.966	36.672	63.229

Conclusioni

In questo capitolo vengono riportate le stringhe di codice usate per effettuare le predizioni sui valori di OEE, pezzi totali prodotti e totale pezzi buoni; viene inoltre effettuato un test sull'accuratezza delle previsioni. Per quanto riguarda l'OEE il codice Python è il seguente.

```
model = pd.read_excel("venv/DATASET.xlsx")
model1 = ['TEMPO OPERATIVO', 'Mese', 'Giorno', 'GUASTI - RISOLTI PM', 'OEE']
model = model[model1]
model1 = model.drop([0, 2, 4], axis=0)
X = model.drop('OEE', axis=1)
y = model['OEE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
rfr = RandomForestRegressor(n_estimators=153, min_samples_split=6, min_samples_leaf=1,
                            max_depth=560, random_state=0)
rfr.fit(X_train, y_train)
dati_input = [[480, 1, 2, 0], [385, 1, 2, 0], [306, 1, 3, 174]]
prediction = rfr.predict(dati_input)
print(prediction)
```

Per effettuare le previsioni ho usato solo le variabili con importanza superiore allo 0,5%, come ricavato dal capitolo precedente: tempo operativo, mese, giorno e guasti risolti PM. Tramite la funzione drop ho eliminato tre righe dal dataset, in modo che il modello, durante l'apprendimento, non potesse apprendere questi valori e soffrire quindi di overfitting.
Ho poi creato la variabile dati_input contenente i valori degli input appena tolti dal dataset, per poi usarla per la previsione, attraverso la funzione predict. Di seguito vengono riportati i valori predetti, confrontati con il valore corretto.

Valore predetto	1	0.603	0.428
Valore esatto	1	0.649	0.422

Come si può notare la prima previsione è perfetta, la seconda ha un'accuratezza del 93% e la terza del 98,6%.
Il modello, quindi, data una qualunque combinazione di dati in ingresso, riesce ad offrire delle previsioni con un'elevata affidabilità, grazie alla corretta ottimizzazione degli iperparametri.
Lo stesso procedimento si può ripetere per il numero totale di pezzi prodotti, di seguito il codice utilizzato e i risultati predetti.

```
model = pd.read_excel("venv/DATASET.xlsx")
model1 = ['TEMPO OPERATIVO', 'Mese', 'MANUTENZIONE PM PROGRAMMATA',
          'MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE', 'Pezzi totali prodotti']
model = model[model1]
model1 = model.drop([0, 2, 4], axis=0)
X = model.drop('Pezzi totali prodotti', axis=1)
y = model['Pezzi totali prodotti']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
rfr = RandomForestRegressor(n_estimators=126, min_samples_split=3, min_samples_leaf=2,
                            max_depth=210, random_state=0)
rfr.fit(X_train, y_train)
dati_input = [[480, 1, 0, 0], [385, 1, 0, 0], [306, 1, 0, 0]]
prediction = rfr.predict(dati_input)
print(prediction)
```

Valore predetto	1798	1268.9	1087.7
Valore esatto	1798	1344	1082

La prima previsione, di nuovo, è perfetta, la seconda del 94,4% e la terza del 99,5%. Anche in questo caso il modello approssima molto bene la realtà.

```
model = pd.read_excel("venv/DATASET.xlsx")
model1 = ['TEMPO OPERATIVO', 'Mese', 'Giorno', 'MICROFERMATE',
          'MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE', 'TOTALE PEZZI BUONI']
model = model[model1]
model1 = model.drop([0, 2, 4], axis=0)
X = model.drop('TOTALE PEZZI BUONI', axis=1)
y = model['TOTALE PEZZI BUONI']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
rfr = RandomForestRegressor(n_estimators=318, min_samples_split=9, min_samples_leaf=2,
                            max_depth=38, random_state=0)
rfr.fit(X_train, y_train)
dati_input = [[480, 1, 0, 0, 0], [385, 1, 0, 20, 0], [306, 1, 0, 0, 0]]
prediction = rfr.predict(dati_input)
print(prediction)
```

Valore predetto	1798	1072.9	791.7
Valore esatto	1798	1150	760

Anche nel caso dei pezzi buoni in totale la prima previsione ha un'accuratezza del 100%, la seconda del 93,3% e la terza del 96%.

In conclusione si può quindi affermare che il modello prodotto è perfettamente funzionante, con una computazione rapida e dai risultati molto accurati.
Volendo poi estendere le previsioni alle altre variabili presenti nel dataset, come FTQ, % scarto, disponibilità e performance, si può ripetere il procedimento, ottimizzando gli iperparametri per la costruzione della random forest ottenendo un modello che rispecchi bene la realtà.

Sitografia

https://www.udemy.com/course/machine-learning-pratico

https://en.wikipedia.org/wiki/Machine_learning

http://www.andreaminini.com/ai/machine-learning/apprendimento-supervisionato

http://www.andreaminini.com/ai/machine-learning/apprendimento-senza-supervisione

https://it.wikipedia.org/wiki/Albero_di_decisione

https://towardsdatascience.com/understanding-random-forest-58381e0602d2

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6

https://www.magnetimarelli.com/it/azienda/la-nostra-storia

https://dialog.it/calcolo-efficienza-produttiva/#:~:text=Il%20risultato%20conduceva%20alla%20seguente,in%20molti%20moderni%20contesti%20produttivi

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://www.python.it/

https://it.wikipedia.org/wiki/Python