UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

Confronto e analisi di misure di feature importance per spiegare approcci machine learning per il riconoscimento della qualità della produzione industriale

Relatori:

Ing. Antonio Luca Alfeo

Prof. Mario G.C.A. Cimino

Candidato:

Cristina Maria Rita Lombardo

ANNO ACCADEMICO 2022/2023

Sommario

L'Industria 4.0 rappresenta una trasformazione radicale nei processi di produzione e gestione aziendale, grazie all'integrazione di tecnologie digitali avanzate. Essa mira a creare un ambiente di produzione intelligente e connesso, in cui le macchine, i sistemi e le persone possono comunicare e collaborare in modo più efficiente.

In questo contesto, l'Intelligenza Artificiale e l'IA spiegabile (XAI) giocano un ruolo cruciale nel migliorare la qualità dei prodotti industriali e ottimizzare i processi produttivi. La XAI è un campo di ricerca che si occupa di sviluppare tecniche e modelli di IA che siano in grado di spiegare le loro decisioni in modo chiaro e comprensibile. Questo può essere fatto attraverso l'uso di tecniche di visualizzazione dei dati, di interpretazione dei modelli e di spiegazione delle decisioni. XAI può aiutare a garantire che i modelli di IA siano affidabili, trasparenti e comprensibili, consentendo un utilizzo sicuro e responsabile. Questo è fondamentale per l'Industria 4.0, dove l'IA è una risorsa preziosa per ottimizzare i processi produttivi e migliorare la qualità dei prodotti e dei servizi.

L'obiettivo di questa tesi è stato quello di applicare i principi dell'Intelligenza Artificiale spiegabile al caso d'uso di un'azienda specializzata nella produzione di macchinari per la lavorazione della carta. La ricerca si è concentrata sull'analisi della qualità dei prodotti creati con macchinari industriali attraverso l'implementazione di algoritmi di machine learning per estrarre informazioni dai dati. In particolare, con i dati forniti dall'azienda sono stati addestrati, con diverse parametrizzazioni, due distinti modelli tramite *MLPClassifier* e *RandomForestClassifier*, in modo da valutare quale dei due fosse più preciso. Attraverso algoritmi di calcolo delle *feature importances* è stato inoltre possibile ottenere un'analisi più approfondita dei dati e della loro influenza sulle decisioni del modello.

Grazie all'utilizzo di tecniche dell'Intelligenza Artificiale spiegabile, quindi, è stato possibile ottenere una maggiore comprensione dei dati e dei processi interni di una *black box* quale è un sistema di IA. I risultati delle previsioni dei modelli di XAI risultano maggiormente significativi poiché è possibile seguire il processo decisionale che ha portato la macchina a sceglierli.

Indice

Capitolo 1

Introduzione

L'idea alla base dell'Intelligenza Artificiale è quella di riuscire a sviluppare delle macchine (hardware o software) dotate di capacità di apprendimento e adattamento ispirate al modello comportamentale degli esseri umani. Oggi l'IA è diventata una forza trainante nella società moderna, trasformando in modo significativo la vita quotidiana e il modo di lavorare. Questa tecnologia trova applicazioni in una grande varietà di settori come la produzione, la logistica, la finanza e la sanità, migliorando allo stesso tempo efficienza e produttività e riducendo i costi. Nell'ottica di una sempre maggiore diffusione delle tecnologie dell'Intelligenza Artificiale, diventa necessario trovare un metodo consolidato per rendere comprensibili i processi decisionali delle macchine IA, anche (e soprattutto) a un pubblico meno familiare al settore informatico. Questa volontà si scontra con il problema della *black box*: la sostanziale opacità dei processi interni degli algoritmi di machine learning. Questa struttura, paragonata appunto a una scatola nera che non permette la visibilità verso l'esterno, diventa un problema rilevante per le applicazioni dell'IA che comportano decisioni ad alto rischio, poiché compromette la possibilità di spiegare le suddette decisioni. È importante sapere quando un modello avrà successo e quando fallirà, perché il modello sta facendo alcune determinate predizioni e fino a che punto possono essere considerate affidabili.

A questa esigenza risponde la XAI (*Explainable Artificial Intelligence*), definita da IBM [1] come "l'insieme di processi e metodi che consentono agli utenti umani di comprendere e considerare affidabili i risultati e gli output generati mediante algoritmi di machine learning". La XAI, attraverso la spiegazione del funzionamento di un sistema automatizzato, permette agli utenti di capire il perché di una determinata scelta e come ogni step intermedio ha influenzato il risultato finale. I vantaggi dell'utilizzo di un modello chiaro sono molteplici e comprendono la riduzione dei rischi dovuti a un utilizzo impreciso o scorretto delle macchine, un aumento della fiducia e dell'adozione da parte degli utenti e la possibilità di ottenere informazioni utili

sui dati osservando le decisioni che il sistema prende su di essi. La XAI permette quindi di affiancare alla *black box* una spiegazione dei suoi output e come questi sono correlati agli input.

Nel'articolo intitolato "Why Should I Trust You? Explaining the Predictions of Any Classifier" [5], gli autori hanno proposto il metodo XAI più utilizzato chiamato *Local Interpretable Model-agnostic Explanations* (LIME). All'interno del saggio, l'applicazione di LIME su immagini che classificano un husky come un lupo fornisce importanti informazioni sul perché la spiegabilità è fondamentale per prevenire un apprendimento inadeguato. L'articolo dimostra come un modello abbia appreso l'esistenza della neve nell'immagine come fattore importante per classificare l'animale come lupo. Tali spiegazioni possono facilmente aiutarci a identificare l'apprendimento difettoso e, a loro volta, aiutare a prevenire la distorsione dei dati e della modellizzazione.

In un contesto di progressiva espansione e sviluppo delle tecnologie di Intelligenza Artificiale, anche il settore industriale si vede protagonista di importanti innovazioni, sia a livello di produzione sia di processi operativi, portando in particolare l'industria 4.0 a un livello superiore. L'industria 4.0, o quarta rivoluzione industriale, si basa sull'integrazione nei processi produttivi di tecnologie digitali avanzate in grado di creare un ambiente di produzione intelligente. L'IA è una componente chiave di questa trasformazione, poiché offre la capacità di analizzare grandi quantità di dati in tempo reale, identificare modelli e tendenze, e prendere decisioni autonome e ottimizzate. Ciò consente alle aziende di ottimizzare i processi di produzione, ridurre i costi, migliorare la qualità dei prodotti e dei servizi, e rispondere in modo più rapido e flessibile alle esigenze del mercato. Il report "Industrial AI and AIoT Market 2021-2026" di IoT Analytics [3] riporta che il tasso di adozione di sistemi intelligenti negli ambienti industriali è del 31%. La maggior parte delle applicazioni di intelligenza artificiale in ambito industriale riguardano la manutenzione predittiva, il controllo predittivo della qualità, l'uso della visione artificiale per il rilevamento dei guasti, la pianificazione e l'ottimizzazione della produzione.

È chiaro che per una corretta integrazione dei sistemi di Intelligenza Artificiale nell'ambito industriale, le azioni e i risultati di questi sistemi devono essere comprensibili e spiegabili agli operatori. L'implementazione della XAI risulta essenziale per facilitare questo processo.

Capitolo 2

Related works

L'Intelligenza Artificiale è una risorsa importante dell'industria 4.0. Le attuali scoperte nell'ambito dell'apprendimento automatico consentono un cambiamento qualitativo all'interno dei processi, delle applicazioni, dei sistemi e dei prodotti industriali. L'IA, ad esempio, può essere implementata per prevedere lo stato di salute di un macchinario prima della sua definitiva rottura (la cosiddetta "manutenzione predittiva") o per indicare e riconoscere la qualità di un prodotto in fase di lavorazione basandosi sui parametri del processo produttivo. Tuttavia, esiste un'importante sfida legata alla comprensibilità (e, quindi, alla fiducia) delle decisioni prese dai modelli di IA. Quando una previsione può essere considerata attendibile e inserita nei processi decisionali industriali?

Nell'articolo "From Artificial Intelligence to Explainable Artificial Intelligence in Industry 4.0: A Survey on What, How, and Where" [2], gli autori chiariscono il ruolo dell'IA nell'industria 4.0, illustrando quali tecnologie dell'Intelligenza Artificiale possono essere sfruttate in questo ambito e in che modo. L'indagine punta a presentare in modo generico gli approcci esistenti per l'applicazione di algoritmi di IA all'industria 4.0, per poi chiarire l'importanza dell'integrazione della XAI nel campo. Inoltre, vengono esposte alcune complicazioni nell'applicazione di questi metodi: oltre alla necessità di utilizzare dispositivi con capacità computazionali notevoli, esiste anche la difficoltà nel bilanciare la comprensibilità dei metodi decisionali dei modelli con la loro capacità nel (provare a) raggiungere un livello di precisione assimilabile a quello dell'essere umano. Inoltre, le tecnologie di IA hanno spesso bisogno di fasi di ottimizzazione degli iperparametri e ingenti quantità di dati sui quali addestrarsi. Diventa indispensabile l'utilizzo di un'IA spiegabile che supporti i risultati degli algoritmi con spiegazioni del modello interno del sistema in modo che gli esperti del settore possano convalidarli per includerli nei processi decisionali, e allo stesso tempo mantenga l'efficacia dei complessi modelli di machine learning.

Un altro problema può sorgere da un apprendimento incorretto del modello, cau-

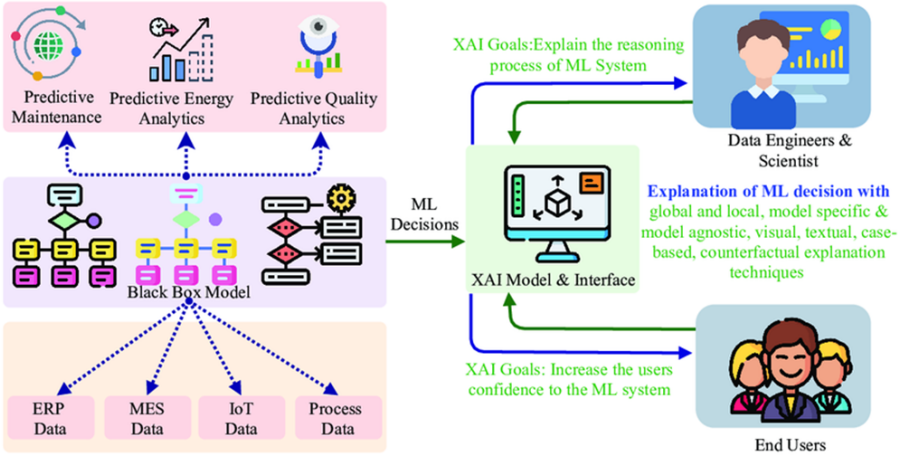

Figura 2.1: Ruolo dellExplainable Artificial Intelligence nell'industria 4.0

sato da una mancanza di dati sufficienti o dalla scarsa qualità di questi. Data la natura del modello, assimilabile a una *black box*, la comprensione degli output del sistema così addestrato diventa ardua e, se ignorato, il problema può portare predizioni ingannevoli che possono avere conseguenze negative sugli obiettivi del sistema di IA. L'utilizzo di modelli basati su XAI può aiutare a comprendere meglio questo tipo di errori, sottolineando l'influenza che determinati dati hanno sui risultati del modello. Secondo lo studio "A Survey of Explainable Artificial Intelligence for Smart Cities" [4]:

"Nell'industria intelligente (*smart industries*), la produzione si basa sempre di più su decisioni fondate sui dati per migliorare gli attuali processi di produzione, e le macchine più recenti vengono utilizzate per risolvere problemi complessi. Le *smart* industries offrono soluzioni sempre migliori per la previsione dei guasti alle macchine e della qualità dei prodotti e per ottenere suggerimenti per risparmiare tempo e costi. Introducendo la tecnologia XAI, l'industria manifatturiera può essere rafforzata e creare un ecosistema affidabile e in continuo sviluppo".

La crescente popolarità delle fabbriche intelligenti e dell'Industria 4.0 ha reso possibile la raccolta di grandi quantità di dati dalle fasi di produzione. Pertanto, i metodi di apprendimento automatico supervisionati, come la classificazione, possono prevedere in modo affidabile la qualità di conformità del prodotto utilizzando questi dati. L'eliminazione delle incertezze attraverso previsioni accurate offre vantaggi significativi in qualsiasi fase della catena di fornitura: una conoscenza tempestiva della qualità dei lotti di prodotto può far risparmiare sui costi associati ai richiami, all'imballaggio e al trasporto.

Lo studio trattato in questa tesi mira a comprendere il comportamento di una macchina addestrata con diversi algoritmi di machine learning attraverso l'uso di modelli di XAI. L'obiettivo è quello di scoprire come migliorare la precisione del modello nel prevedere la qualità dei prodotti generati da alcuni macchinari industriali, e di comprendere in dettaglio i processi decisionali della macchina stessa attraverso l'analisi dei dati e il loro contributo nel risultato finale.

Capitolo 3

Design e implementazione

3.1 Design

Seguendo gli obiettivi di questa tesi, in questa sezione verranno messi in chiaro i procedimenti che sono stati seguiti nell'ottica di realizzare un software in grado di addestrare più tipi di modelli, sperimentare diverse parametrizzazioni e valutare il peso dei dati nelle decisioni della macchina.

3.1.1 Classificazione

Gli algoritmi di classificazione sono metodi utilizzati per assegnare un'etichetta o una classe a un determinato insieme di dati in modo che questi possano essere analizzati e utilizzati per prendere decisioni informate. In particolare, ci si è concentrati sugli algoritmi di classificazione supervisionata: questi richiedono un insieme di dati di addestramento etichettati, cioè dati in cui è già nota l'etichetta o la classe a cui appartengono. L'algoritmo utilizza questi dati per imparare a classificare nuovi dati non ancora etichettati.

3.1.2 Modelli

Nell'ambito dell'Intelligenza Artificiale, un modello è una rappresentazione matematica di un sistema, un processo o un fenomeno che consente di estrarre informazioni utili dai dati e di creare sistemi intelligenti che possano fare previsioni, prendere decisioni e adattarsi a nuove situazioni.

Il primo modello utilizzato è MLP che sta per "*Multi-Layer Perceptron*" ed è un tipo di rete neurale artificiale (ANN). Le ANN sono composte da livelli di nodi che contengono un livello di input, uno o più livelli nascosti e un livello di output. Ciascun nodo, o neurone artificiale, si connette ad un altro e ha un peso e una soglia associati. Se l'output di qualsiasi singolo nodo è al di sopra del valore di soglia

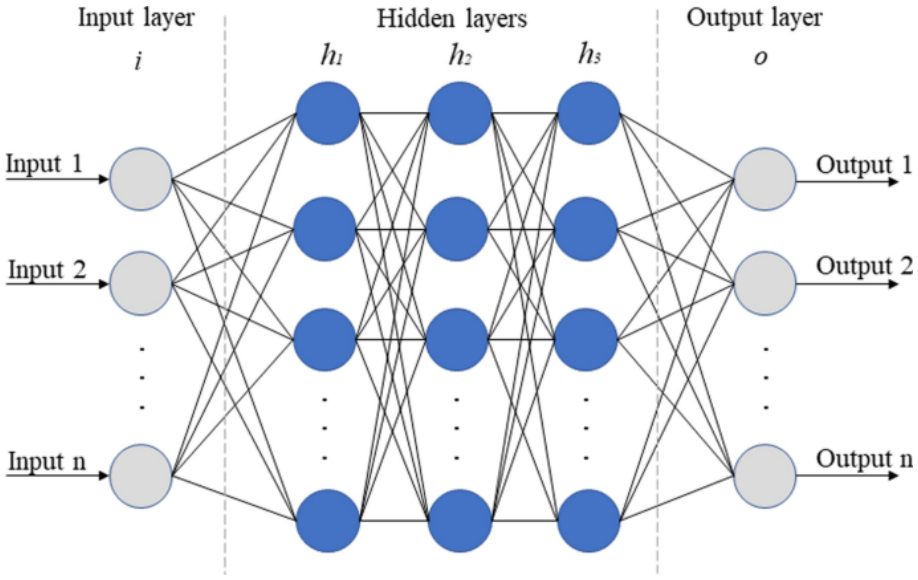

Figura 3.1: Schema riassuntivo del funzionamento di MLP

specificato, tale nodo viene attivato, inviando i dati al successivo livello della rete. Le reti neurali fanno affidamento sui dati di addestramento per imparare e migliorare la loro accuratezza nel tempo. Questo addestramento coinvolge l'aggiustamento dei pesi delle connessioni in modo che l'errore tra l'output previsto e l'output desiderato sia ridotto. MLP ha anche alcune limitazioni, come la tendenza al sovradattamento (*overfitting*) e la necessità di una quantità significativa di dati di addestramento.

Si ha *overfitting* quando l'apprendimento è stato effettuato troppo a lungo o quando c'è uno scarso numero di esempi di allenamento. In questi casi il modello potrebbe adattarsi a caratteristiche che sono specifiche solo dei dati di allenamento, ma che non hanno riscontro nel resto dei casi, quindi quando il miglioramento delle prestazioni del modello (cioè la capacità di adattarsi/prevedere) sui dati di allenamento non implica un miglioramento delle prestazioni sui dati nuovi.

Il secondo modello utilizzato è il *Random Forest*, che è un algoritmo di apprendimento automatico che appartiene alla categoria degli algoritmi di "*ensemble*". Questi combinano diversi modelli per migliorare le prestazioni complessive del sistema. In particolare, *Random Forest* utilizza un insieme di alberi decisionali (*Decision Trees*) per fare previsioni e combina queste previsioni per ottenere una decisione finale. Ogni albero decisionale è costruito su un sottoinsieme casuale dei dati di addestramento e utilizza un sottoinsieme casuale delle caratteristiche per fare previsioni. Questo particolare processo di campionamento rende gli alberi indipendenti tra loro e riduce il rischio di *overfitting*. *Random Forest* è particolarmente utile per

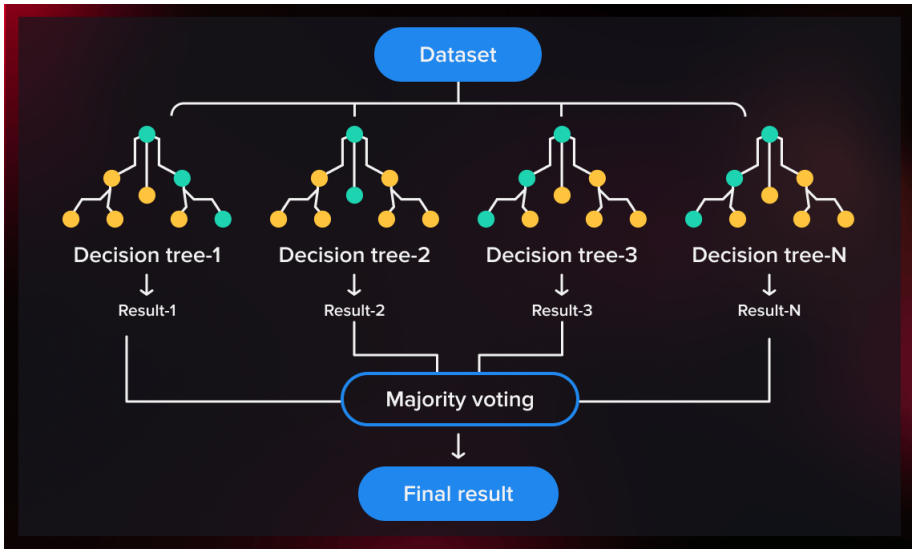

Figura 3.2: Schema riassuntivo del funzionamento di Random Forest

problemi con un grande numero di caratteristiche e dati rumorosi, poiché è in grado di gestire entrambi i casi in modo efficace. Tuttavia, ha anche alcune limitazioni, come la tendenza a sovrastimare l'importanza delle caratteristiche e la necessità di un'accurata ottimizzazione dei parametri.

3.1.3 Parametrizzazioni

Gli iperparametri controllano la precisione del modello e sono configurati a priori da chi sviluppa il software. È compito suo, infatti, trovare un set di iperparametri che massimizzi le performance del modello. L'ottimizzazione degli iperparametri è un'attività importante nell'apprendimento automatico ed è spesso fatta utilizzando tecniche di ricerca a griglia o di ricerca casuale. La ricerca a griglia (grid search) automatizza la ricerca dei valori ottimali e consiste nel definire una griglia di valori per ogni iperparametro e testare tutte le loro possibili combinazioni: per ognuna, il modello viene addestrato e analizzato utilizzando una metrica di valutazione, come l'accuratezza. Alla fine della grid search, si selezionano i valori di iperparametri che massimizzano la metrica.

Nel caso del modello MLP, sono stati presi in considerazione i seguenti iperparametri:

- *solver*: è responsabile della scelta dell'algoritmo di ottimizzazione utilizzato per addestrare la rete neurale. Possibili valori: 'lbfgs', 'sgd', 'adam'

- *max_iter*: rappresenta il massimo numero di iterazioni. Possibili valori: numeri interi
- *hidden_layer_sizes*: specifica il numero di layer nascosti nella rete e il numero di nodi in ciascun layer. Possibili valori: array di interi di lunghezza n_layer–2
- *learning_rate*: controlla il tasso di apprendimento durante l'addestramento della rete neurale. Possibili valori: 'constant', 'invscaling', 'adaptive'
- *learning_rate_init*: decide il tasso di apprendimento iniziale. Possibili valori: numeri reali

Nel caso del modello Random Forest, sono stati scelti i seguenti iperparametri:

- *n_estimators*: indica il numero di alberi decisionali utilizzati nel modello. Possibili valori: numeri interi
- *max_features*: controlla il numero massimo di feature considerate per trovare il miglior split in ogni nodo dell'albero. Possibili valori: 'none', 'sqrt', 'log2', numeri reali
- *min_samples_leaf*: indica il numero minimo di campioni richiesti in una foglia. Possibili valori: numeri reali
- *max_depth*: controlla la massima profondità degli alberi decisionali. Possibili valori: numeri interi

3.1.4 Feature importances

L'importanza di una caratteristica (*feature importance*) misura l'impatto di questa sulle prestazioni del modello. All'interno della tesi sono state utilizzate diverse tecniche per il calcolo di questi valori.

La prima tecnica è la *Permutation Importance*. Essa misura il contributo di ciascuna feature calcolando l'accuratezza di un modello prima e dopo aver scambiato in modo casuale i valori della caratteristica stessa. Minore è l'accuratezza (e quindi maggiore è l'errore di predizione del modello) dopo una permutazione dei valori, maggiore è il peso di quella feature per quel particolare modello. *Permutation Importance* è molto sensibile alla casualità delle permutazioni. Questo può essere risolto calcolando l'importanza come la media dei valori dati da un numero maggiore di permutazioni.

La seconda tecnica si basa sul calcolo del *Gini Index*. Esso è una misura di impurità in un albero decisionale (quindi non può essere utilizzato per qualsiasi tipo di modello). Più basso è il valore dell'indice in un nodo, maggiore è la sua purezza, il che significa che i dati in quel nodo sono omogenei rispetto alla classe di output. Al contrario, un indice più alto indica maggiore eterogeneità nei dati. Per una determinata feature, maggiore è la somma delle riduzioni del *Gini Index* in tutti gli split dell'albero che la riguardano, maggiore sarà la sua importanza all'interno del modello. Nel caso dell'algoritmo di classificazione *Random Forest* vengono considerati gli split in tutti gli alberi. Questa misura viene calcolata durante l'addestramento e al termine di questo viene messa a disposizione nella variabile `model.feature_importances_`. Questo metodo ha lo svantaggio di essere notevolmente parziale, incrementando l'importanza di features con alta cardinalità a dispetto di quelle con bassa cardinalità, tra le quali rientrano tipicamente le variabili categoriche, fortemente presenti nel dataset in esame.

Un terzo metodo per calcolare le feature importances è attraverso i valori di Shapley. SHAP ("*SHapley Additive exPlanations*") è un framework per l'interpretazione dei modelli di machine learning che si basa sui valori di Shapley. Un valore di Shapley è un concetto di soluzione utilizzato per assegnare una ricompensa a ciascun giocatore in un gioco cooperativo in base al contributo di quel giocatore al risultato della partita. Nell'ambito dell'interpretazione dei modelli, i valori di Shapley rappresentano l'impatto di ciascuna feature nelle predizioni del modello stesso. Entrando nel dettaglio, per calcolare il contributo di una feature a una coalizione (ovvero a una combinazione di caratteristiche), viene calcolata la differenza tra l'accuratezza del modello in sua presenza e l'accuratezza in sua assenza. Il risultato è detto "contributo marginale" della feature alla coalizione. Questa idea può essere estesa poi al caso di un dataset con tante caratteristiche. Infatti, dato un set di n feature, esistono 2^n coalizioni possibili. Il valore di Shapley di una caratteristica corrisponde alla media di tutti i contributi marginali ottenuti da ognuna delle possibili combinazioni di feature in cui è inclusa anche quella iniziale. Questo metodo ha il grosso vantaggio di valutare l'importanza di ogni caratteristica anche in relazione alle altre features, e non in modo individuale come fanno i precedenti algoritmi. Per dataset che presentano feature correlate tra loro, questa variazione può essere determinante per ottenere valori di importanza sensati.

3.2 Implementazione

Il software è stato implementato in Python grazie all'utilizzo di alcune librerie: NumPy, Matplotlib, Sklearn, Pandas, SHAP e Seaborn.

Per permettere una gestione semplice dei dati è stata creata la classe *Dataset*. Segue una spiegazione dei suoi metodi:

- `__init__(self)`. Descrizione: legge il file CSV contenente il dataset iniziale e rimuove la prima colonna
- `split_data(self)`. Descrizione: divide i dati in set di addestramento e di test tramite la funzione `train_test_split` della libreria scikit-learn
- `encode_data(self)`. Descrizione: utilizza la funzione *OrdinalEncoder* per codificare i dati categorici del dataset in numeri interi
- `save_data(self)`. Descrizione: salva i dati di addestramento e di test in file CSV
- `recover_data(self)`. Descrizione: recupera i dati di addestramento e di test (già suddivisi) dai corrispettivi file CSV
- `scale_data(self)`. Descrizione: fa lo scaling dei dati utilizzando la funzione *MinMaxScaler*
- `save_params(self, params)`. Descrizione: salva i parametri migliori del modello in un file JSON
- `load_params(self)`. Descrizione: carica i parametri migliori del modello in un file JSON

3.3 Use case

Attori:

- data scientist: professionista che sviluppa e implementa algoritmi di machine learning per analizzare i dati
- responsabile di produzione: professionista che utilizza i risultati dell'analisi per ottimizzare i processi produttivi

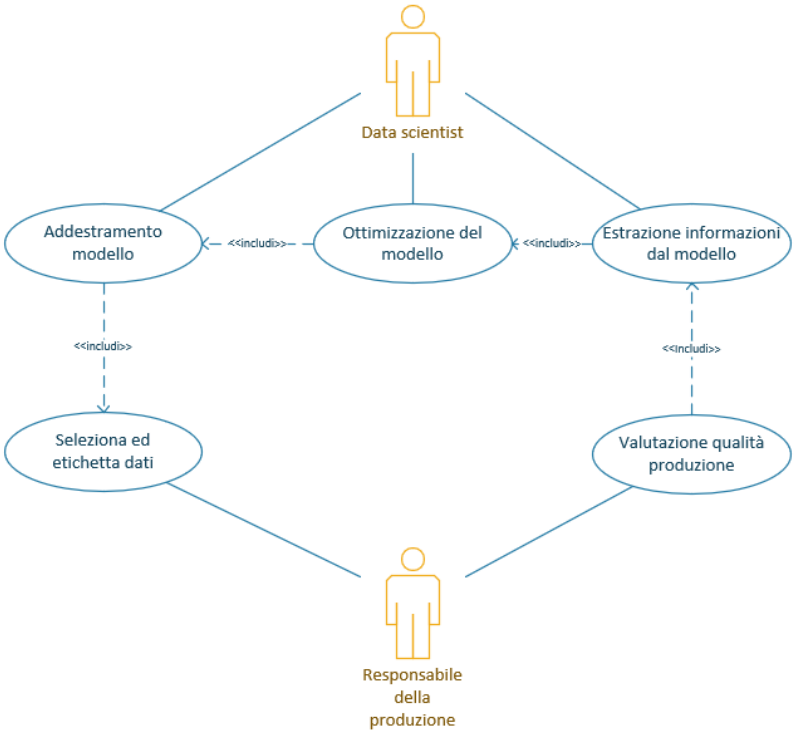

Figura 3.3: Diagramma degli use case

Casi d'uso per il data scientist:

- addestramento modello: addestra i modelli su dati precedentemente valutati da esperti in modo che la macchina impari a valutare in modo autonomo i dati che non sono ancora stati classificati
- ottimizzazione del modello: in base al tipo di dataset, analizza il comportamento del modello con diverse parametrizzazioni e sceglie quella che massimizza le sue prestazioni
- estrazione informazioni dal modello: ricava informazioni legate ai dati sul comportamento del modello e i suoi processi decisionali

Casi d'uso per il responsabile della produzione:

- raccolta dati: colleziona e cataloga i dati generati durante la fase di lavorazione dai macchinari industriali
- valutazione qualità produzione: grazie ai dati acquisiti diventa consapevole di quali siano i fattori che maggiormente influenzano i risultati della sua attività e prende decisioni legate alla pianificazione e al controllo del processo produttivo

Capitolo 4

Case study

Il caso di studio di questa tesi riguarda l'analisi di dati con l'obiettivo di riconoscere la qualità della carta in uscita da un processo produttivo industriale a partire da misurazioni effettuate durante la lavorazione. Il dataset è stato fornito dalla "Fabio Perini S.p.A", un'azienda specializzata nella progettazione e produzione di macchinari per la lavorazione della carta tissue, e contiene informazioni riguardo il livello di predizione della qualità del prodotto.

4.1 Il dataset

Il dataset "BULK" consiste di 440 righe e 18 features. L'etichetta è rappresentata dalla colonna *media_bulk_cat* ed assume valori compresi tra 1 e 3 che indicano rispettivamente un livello di qualità della carta basso, medio e alto. All'interno del dataset non sono presenti valori nulli poiché i dati hanno già subito una prima fase di *preprocessing* nella quale è stata eseguita una scrematura per eliminare righe ridondanti o con troppi valori mancanti e feature poco significative.

Il dataset contiene i seguenti tipi di informazioni:

- Bulk: la caratteristica considerata per definire la qualità del prodotto finale realizzato dalla macchina
- Res Lon e Res Lat: la resistenza della carta grezza in senso longitudinale e latitudinale, misurata in Newton per metro
- Allungamento: la percentuale di allungamento della carta grezza, in senso longitudinale, se bagnata
- Grammatura della carta: il rapporto tra il peso della carta e la sua superficie, misurato in grammi per metro quadrato

- Durezza del rullo di gomma: la durezza del rullo di gomma utilizzato per imprimere il motivo sulla carta
- Ribobinatrice ed embosser: identificatore univoco del modello di questi componenti
- Motivo del rullo inferiore e superiore: un identificatore univoco del motivo del disegno che caratterizza i rulli dell'embosser
- Struttura della carta: un valore booleano che indica se la carta grezza è costituita da carta normale o strutturata
- Tipologia Prodotto: il prodotto oggetto di realizzazione
- Strati di tessuto: il numero di strati
- Allungamento Lat: la percentuale di allungamento della carta grezza, in senso latitudinale, se bagnata
- Rapporto di allungamento a secco: rapporto tra la resistenza della carta grezza e l'allungamento a secco in senso longitudinale e latitudinale

4.2 Preprocessing

Addestrare i modelli predittivi richiede che i dati siano in un formato adeguato. Per poter lavorare con il dataset sopra descritto è stata svolta una fase di *preprocessing* composta da due operazioni. Nella prima è stato necessario codificare, tramite la classe *OrdinalEncoder* della libreria *scikit-learn*, i dati categorici presenti nel dataset in numeri interi, in modo che l'algoritmo di apprendimento potesse utilizzarli. Con l'*Ordinal Encoding* (o codifica ordinale) assegniamo ai vari livelli della feature un valore numerico sequenziale. Questa tecnica è utile poichè mantiene l'ordine progressivo dei livelli della variabile ordinale e permette di codificare la feature attraverso la creazione di una sola variabile.

Per rendere i dati di diverse feature comparabili tra loro, la seconda operazione effettuata è stata lo scaling dei dati di addestramento e di test utilizzando la classe *MinMaxScaler* della libreria *scikit-learn*. In questo modo le caratteristiche di input sono state ridimensionate in base ai valori minimi e massimi dei dati di addestramento.

Il normalizzatore *MinMax* ridimensiona in modo lineare ogni feature all'intervallo [0,1] (intervallo di default). I valori nella colonna vengono trasformati usando la seguente formula:

$$x_{scalato} = \frac{x - min(x)}{max(x) - min(x)}$$

Capitolo 5

Risultati sperimentali

5.1 Metriche e grafici

Accuracy score: è una metrica di valutazione che misura la percentuale di predizioni corrette fatte dal modello su un insieme di dati di test. Ovvero indica quanto il modello è in grado di classificare correttamente le istanze del dataset di test. L'interpretazione del suo valore è semplice: maggiore è il numero di previsioni corrette, maggiore sarà il punteggio di precisione ottenuto. Il miglior valore possibile è 1 (se un modello ha ottenuto tutte le previsioni corrette) e il peggiore è 0 (se un modello non ha effettuato una singola previsione corretta). In generale, nei casi in cui non sono previste decisioni rischiose, viene considerato un *accuracy score* > 0.9 come un punteggio eccellente, $>$ 0,7 come un punteggio buono e qualsiasi altro valore come un punteggio scarso. L'*accuracy* viene calcolata tramite la seguente formula:

$$Accuracy = \frac{Numerodipredizionicorrette}{Numerototaledipredizioni}$$

Grafico di linea: mostra la relazione tra due variabili. In particolare, è stato utilizzato per valutare i cambiamenti dell'accuratezza del modello addestrato con classificatore MLP al variare del numero di iterazioni dell'algoritmo.

Bar plot: rappresenta dati categorici con barre rettangolari di lunghezza variabile in base al valore associato a quella categoria. In questo caso è stato utilizzato per mostrare le *feature importances* di un modello di machine learning, con ciascuna barra che rappresenta una feature e il suo peso nelle decisioni del modello.

Matrice di correlazione: una tabella che viene costruita calcolando il coefficiente di correlazione per ogni coppia di variabili e inserendolo nella cella corrispondente. Può essere utilizzata per determinare quali variabili sono significativamente correlate tra loro e quali sono scarsamente o per nulla correlate. Il coefficiente di correlazione varia da -1 a +1, dove -1 indica una perfetta correlazione negativa, +1 una perfetta

correlazione positiva e 0 significa che non c'è correlazione tra le variabili. Le variabili avranno coefficienti di correlazione positivi elevati se tendono a salire o scendere insieme, e negativi elevati se tendono a salire o scendere in direzioni opposte. Per calcolare il coefficiente di correlazione tra due variabili si utilizza la seguente formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.1}$$

- x_i, y_i: valori di x e y
- $\bar{x}$, $\bar{y}$: valori medi delle variabili x e y

In questo contesto, le variabili corrispondono alle features del dataset: conoscere i legami tra queste può fornire informazioni su come la variazione di una feature può influire su un'altra. Le features con bassi coefficienti di correlazione portano all'addestramento di un modello più stabile e generalizzato, poiché ciascuna di esse fornisce informazioni uniche e indipendenti. Analizzare queste correlazioni può essere utile per identificare feature ridondanti, ridurre i *bias* del modello, aumentare la sua spiegabilità oppure, motivo per il quale è stata utilizzata in questa ricerca, testare la robustezza degli algoritmi di calcolo delle *feature importances* alla correlazione tra caratteristiche.

5.2 Attività svolta

Inizialmente, come già anticipato, è stata svolta una fase di *preprocessing* all'interno della quale i dati categorici sono stati codificati, tramite la classe *OrdinalEncoder* della libreria *scikit-learn*, in numeri interi. È stato inoltre effettuato lo scaling dei dati di addestramento e di test utilizzando la classe *MinMaxScaler* della libreria *scikit-learn*. Dopo aver inizializzato e preparato il dataset, è stato addestrato un *MLPClassifier* (importato dalla libreria *scikit-learn*) utilizzando una ricerca a griglia (*GridSearchCV*, anche questa importata da *scikit-learn*) per trovare i migliori parametri che potessero dare risultati soddisfacenti. Dopo aver addestrato il modello, è stato utilizzato il metodo *predict* del *MLPClassifier* per fare delle previsioni su dati di test e ottenere le etichette previste per questi dati. La funzione *accuracy_score* della libreria *scikit-learn* è stata utilizzata per calcolare il punteggio di accuratezza.

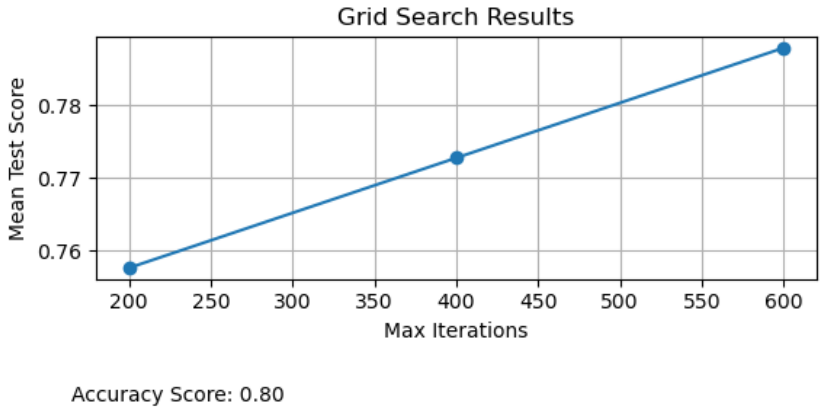

Figura 5.1: Grafico di linea che mostra come il punteggio medio di test cambia con il numero massimo di iterazioni per l'algoritmo di ottimizzazione

Il grafico in Figura 5.1, che rappresenta una singola esecuzione del codice, mostra alcuni parametri di `cv_results_` che è un dizionario che contiene i risultati della ricerca a griglia. Il primo parametro rappresenta il numero massimo di iterazioni per l'algoritmo di ottimizzazione (riportato sull'asse x), mentre l'altro è la media dei punteggi di test per un particolare parametro (riportato sull'asse y). Viene inoltre stampato il punteggio di accuratezza del modello.

I risultati di questa e altre valutazioni condotte sul dataset evidenziano valori di accuratezza intorno allo 0.8. Ciò indica che il modello è in grado di prevedere correttamente la qualità dell'80% dei prodotti nel dataset. Questo primo risultato, per quanto apprezzabile, ha spinto ad indagare maggiormente sulle sue motivazioni, esplorando l'influenza di diversi iperparametri sugli algoritmi. In particolare, per poter valutare anche l'efficacia di un modello come MLP, è stato messo a paragone con *Random Forest*, confrontando come i due classificatori si comportavano sugli stessi dati. In entrambi i casi è stata effettuata una ricerca a griglia con *GridSearchCV* per individuare i migliori valori per ciascun iperparametro descritto precedentemente.

L'analisi è cominciata dal classificatore MLP. Per ogni iperparametro è stato scelto un determinato range da testare. In particolare, per i parametri che hanno un insieme illimitato di valori possibili, sono stati effettuati più tentativi per vedere quali fossero i risultati sul modello addestrato. Nello specifico:

- *max_iter*: sono stati scelti i valori '800', '1000' e '1200' poiché un numero di iterazioni maggiore permette all'algoritmo di convergere, rendendo il modello più stabile

- *hidden_layer_sizes*: sono state scelte le tuple (200, 100, 50, 25, 10), (50, 25) e (100, 100, 50, 25). Questo perché algoritmi di ottimizzazione diversi portano a risultati migliori con un diverso numero di layer nascosti. Il numero di nodi per ogni layer è stato scelto dopo numerosi test. Il loro decremento è utile a limitare la capacità di apprendimento del modello per evitare l'overfitting

- *learning_rate_init*: sono stati scelti i valori '0.01' e '0.001' per evitare che una velocità di apprendimento troppo elevata potesse causare instabilità durante l'addestramento

Per quanto riguarda il classificatore *Random Forest*, invece, sono stati considerati maggiormente significativi per l'addestramento del modello i seguenti range di valori:

- *n_estimators*: sono stati scelti i valori '10', '200' e '1000' per poter testare il modello con un numero di alberi decisionali molto diverso. Un valore più alto tende a ridurre il rischio di overfitting, ma comporta un aumento del tempo di addestramento e delle risorse computazionali necessarie

- *min_samples_leaf*: sono stati scelti i valori '1', '2', '3', '4' e '5' in modo da permettere al modello di adattarsi meglio ai dati di addestramento, rischiando però un maggiore overfitting

- *max_depth*: sono stati scelti i valori '10', '20' e 'None' in modo da poter testare l'algoritmo con alberi di diversa profondità. Un valore più basso limita la capacità del modello di memorizzare i dati di addestramento, rendendolo meno incline all'overfitting. Un valore più alto invece può portare alla creazione di alberi più profondi e complessi, aumentando i tempi di addestramento e il rischio di overfitting.

Di seguito le combinazioni di parametri che hanno permesso di ottenere i migliori risultati di accuratezza insieme a un'analisi delle possibili motivazioni.

MLPClassifier:

- *solver*: 'adam'. Adattando automaticamente il tasso di apprendimento, rende più rapida e precisa la convergenza

- *max_iter*: 1000. Assicura che l'addestramento continui fino alla convergenza del modello

- *hidden_layer_sizes*: (200, 100, 50, 25, 10). Un'architettura relativamente complicata consente al modello di apprendere relazioni complesse tra i dati
- *learning_rate*: 'constant'. Permette un maggiore controllo sull'adattamento del tasso di apprendimento implementato dall'algoritmo 'adam', rendendo stabile il processo di ottimizzazione
- *learning_rate_init*: 0.01. Consente un rapido avvio dell'addestramento, la cui velocità verrà poi adattata gradualmente grazie all'algoritmo 'adam'

Punteggio di accuratezza: 0.863

RandomForestClassifier:

- *n_estimators*: 10. Un valore più alto aumenta eccessivamente la complessità del modello senza migliorare, almeno in questo caso, le sue prestazioni
- *max_features*: 'None'. Fornisce maggiore flessibilità al modello e può essere utile per catturare relazioni complesse tra le variabili
- *min_samples_leaf*: 1. Permette al modello di memorizzare dettagli specifici dei dati di addestramento, aumentando la sua precisione
- *max_depth*: None. Consente agli alberi di crescere fino a quando non riescono a distinguere perfettamente i dati di addestramento

Punteggio di accuratezza: 0.882

Proseguendo nel tentativo di estrapolazione delle informazioni tramite modelli di XAI, il passo successivo è stato quello di calcolare l'importanza di ciascuna feature del dataset attraverso diversi algoritmi che prendono in considerazione fattori differenti per produrre i dati, e poi mettere a confronto i risultati per determinare quanto i metodi si avvicinassero a una conclusione univoca. L'utilità di un'operazione del genere è quella di comprendere come ogni caratteristica del dataset influenzi singolarmente le prestazioni del modello, ovvero su cosa di basa quest'ultimo per prendere le sue decisioni.

Come già anticipato, le prime due tecniche sono la *Permutation Importance* e l'utilizzo del *Gini Index*. Per poter effettuare un confronto tra le due è stato addestrato un modello attraverso il *RandomForestClassifier* e sono stati provati diversi split del dataset attraverso la funzione `train_test_split`, che divide in modo casuale il 75%

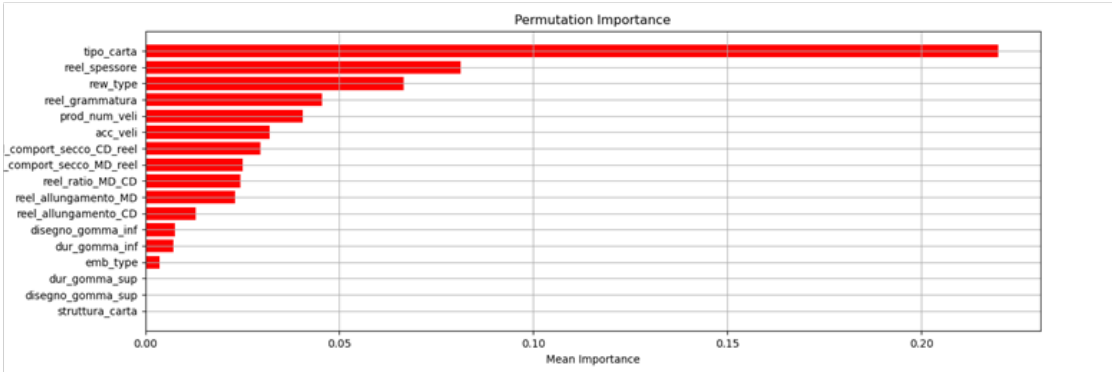

Figura 5.2: Importanza delle feature calcolata con Permutation Importance. Accuracy score del modello: 0.882

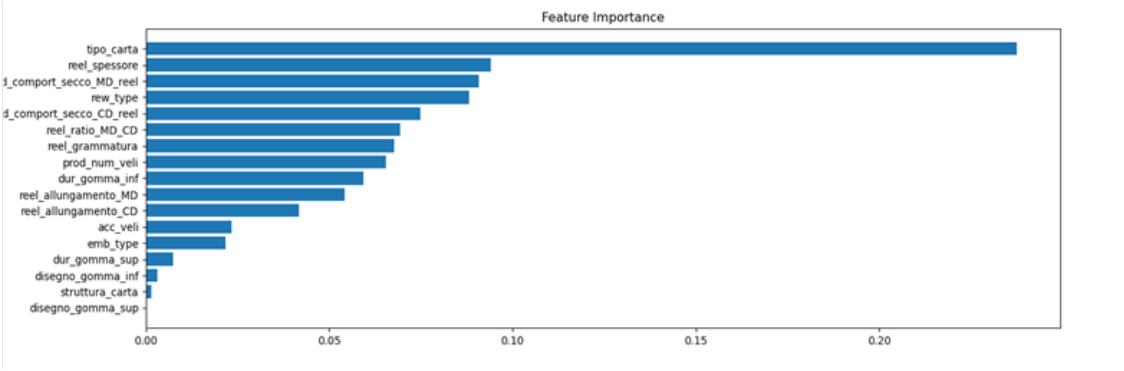

Figura 5.3: Importanza delle feature calcolata tramite Gini Index. Accuracy score del modello: 0.882

e il 25% del dataset rispettivamente nei sottoinsiemi di training e di testing. L'algoritmo di *Permutation Importance* è stato usato sui dati di testing con parametro `n_repeats` pari a 50, in modo da stabilizzare la misura per ogni caratteristica. Sono stati confrontati i valori delle importanze calcolati da *Permutation Importance* e contenuti nella variabile `model.feature_importances_` tramite l'utilizzo di bar plot.

Il passo successivo è stato quello di capire e spiegare le motivazioni dietro i risultati di *Permutation Importance* e la loro dipendenza dall'accuratezza del modello. Inoltre, per poter valutare l'efficacia dell'algoritmo, lo si è messo a confronto con un altro metodo per calcolare le *feature importances* che si basa sui valori di Shapley. Per poter comprendere le problematiche dell'algoritmo di *Permutation Importance* e la sua dipendenza dalla precisione del modello, bisogna considerare due suoi difetti particolarmente rilevanti nel nostro specifico caso:

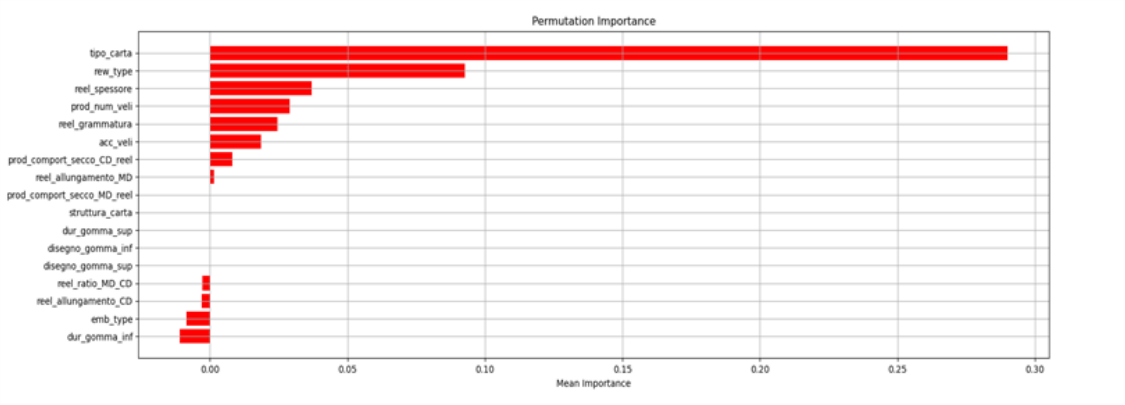

Figura 5.4: Importanza delle feature calcolata con Permutation Importance. Accuracy score del modello: 0.818

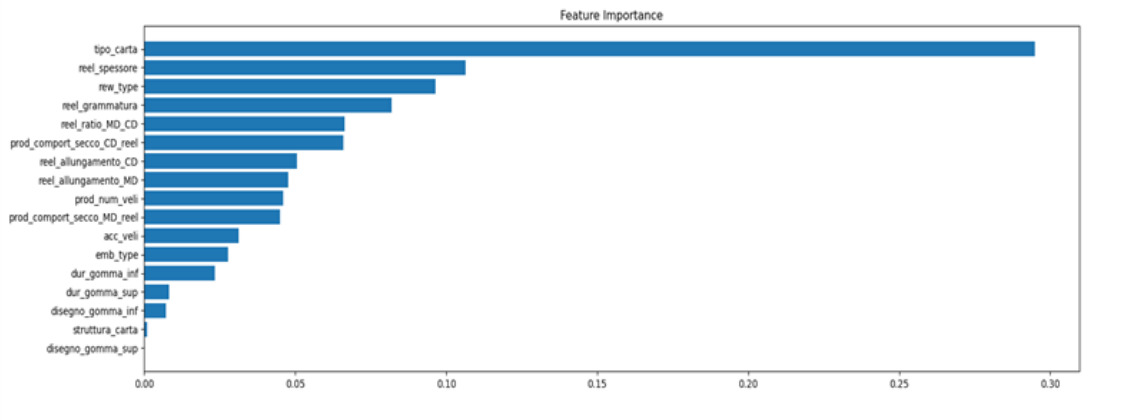

Figura 5.5: Importanza delle feature calcolata tramite Gini Index. Accuracy score del modello: 0.818

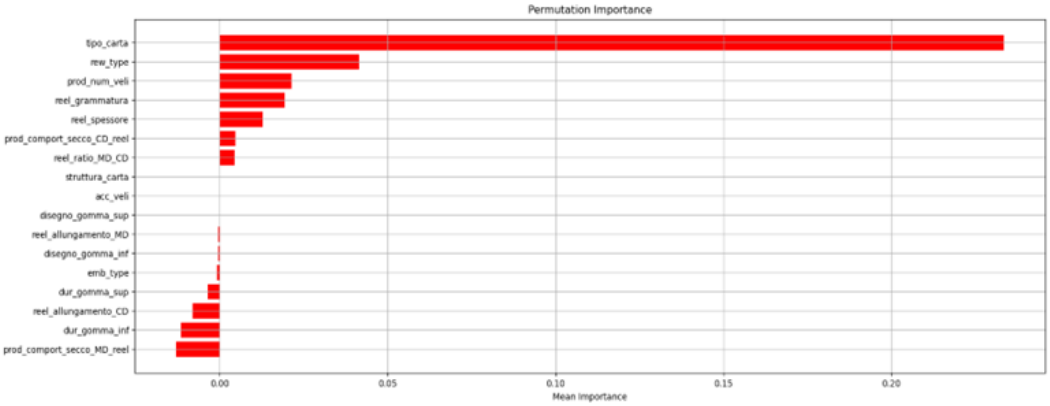

Figura 5.6: Importanza delle feature calcolata con Permutation Importance. Accuracy score del modello: 0.709

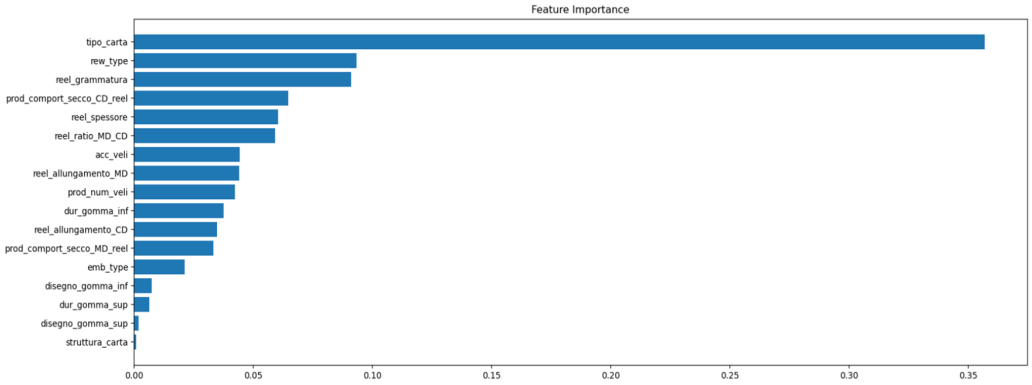

Figura 5.7: Importanza delle feature calcolata tramite Gini Index. Accuracy score del modello: 0.709

- la casualità delle permutazioni: ogni singola permutazione è dettata dalla casualità e può produrre risultati molto differenti tra un'esecuzione e un'altra. In particolare, questa variazione è evidente nel caso di piccoli dataset, come quello preso in esame

- ogni feature viene valutata singolarmente: le permutazioni vengono effettuate su una sola feature alla volta. Questo può essere un problema quando nel dataset sono presenti caratteristiche correlate tra loro. Ad esempio, se due feature sono interdipendenti, permutando i valori di una, la performance del modello potrebbe non cambiare molto perché la seconda fornisce informazioni simili alla prima. In questo caso, l'algoritmo potrebbe sottostimare l'importanza della caratteristica permutata

In generale, quando ci sono variabili correlate, *Permutation Importance* potrebbe non essere in grado di distinguere tra features che forniscono informazioni simili e features che forniscono informazioni uniche. Per valutare la robustezza dell'algoritmo alla correlazione tra features è stata generata una matrice di correlazione sui dati preprocessati per valutare il livello di interdipendenza tra le stesse.

In particolare, sono presenti correlazioni tra le seguenti features:

- *prod_comport_secco_CD_reel* e *prod_comport_secco_MD_reel* (0.84)

- *prod_comport_secco_CD_reel* e *tipo_carta* (0.88)

- *prod_comport_secco_MD_reel* e *tipo_carta* (0.83)

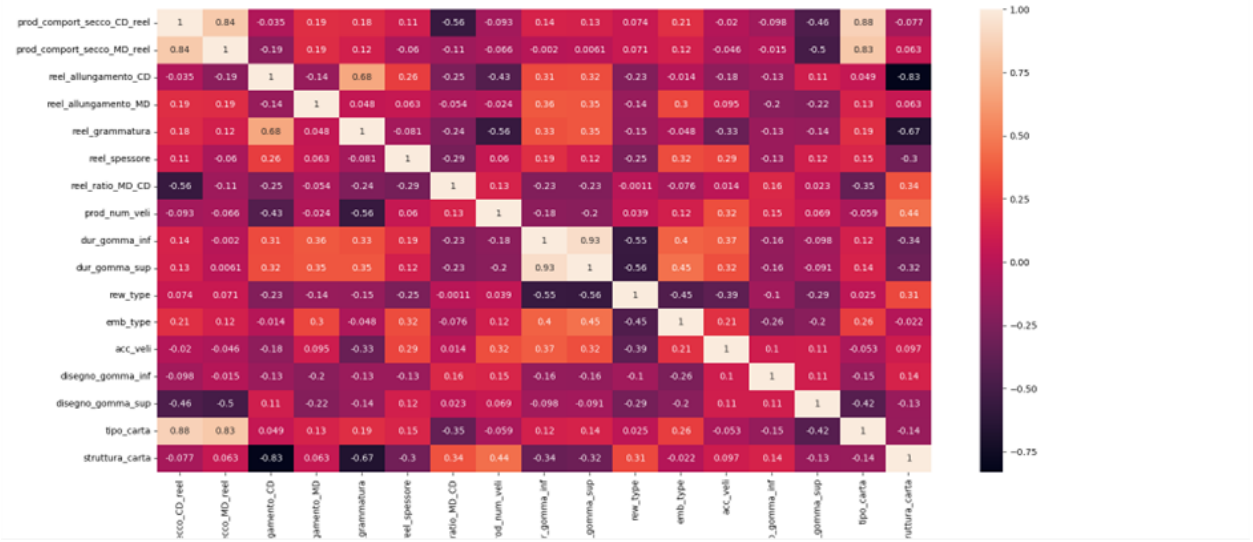

Figura 5.8: Mappa dei coefficienti di correlazione tra ogni coppia di feature del dataset

- *reel_ allungamento_ CD* e *reel_ grammatura* (0.68)
- *dur_ gomma_ inf* e *dur_ gomma_ sup* (0.93)

Successivamente, per poter confrontare *Permutation Importance* con un altro metodo, sono stati calcolati, per diversi split del dataset, i valori di Shapley delle feature tramite la funzione `TreeExplainer`. Questa, messa a disposizione dalla libreria *shap*, utilizza degli algoritmi che spiegano l'output di un modello ad albero (il modello è stato addestrato tramite *Random Forest Classifier*). I seguenti grafici mostrano il confronto, a parità di accuratezza del modello, tra le importanze delle feature calcolate con *Permutation Importance* e quelle calcolate tramite i valori di Shapley sugli stessi split del dataset.

Come si può notare dai grafici, gli algoritmi di calcolo di *Permutation Importance* non riescono a trarre risultati definitivi sulle importanze delle feature correlate: alle caratteristiche con coefficiente di correlazione alto vengono assegnati valori alternativamente alti o bassi, rendendo impossibile l'interpretabilità dei dati. D'altra parte, per quanto anche questi fossero leggermente variabili, i risultati ricavati attraverso i valori di Shapley risultano decisamente più stabili e coerenti da un'iterazione a un'altra. Ciò è dovuto al diverso approccio di questo algoritmo: nel calcolo dell'importanza di una feature non viene semplicemente considerato il suo contributo

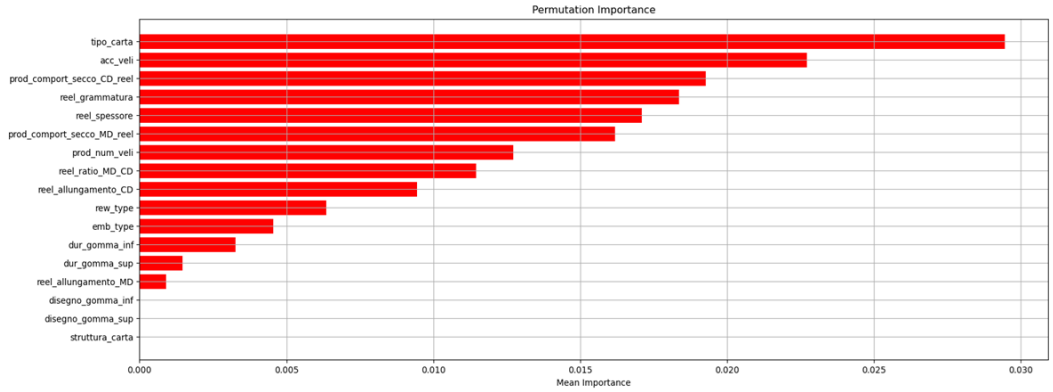

Figura 5.9: Importanza delle feature calcolata con Permutation Importance. Accuracy score del modello: 0.845

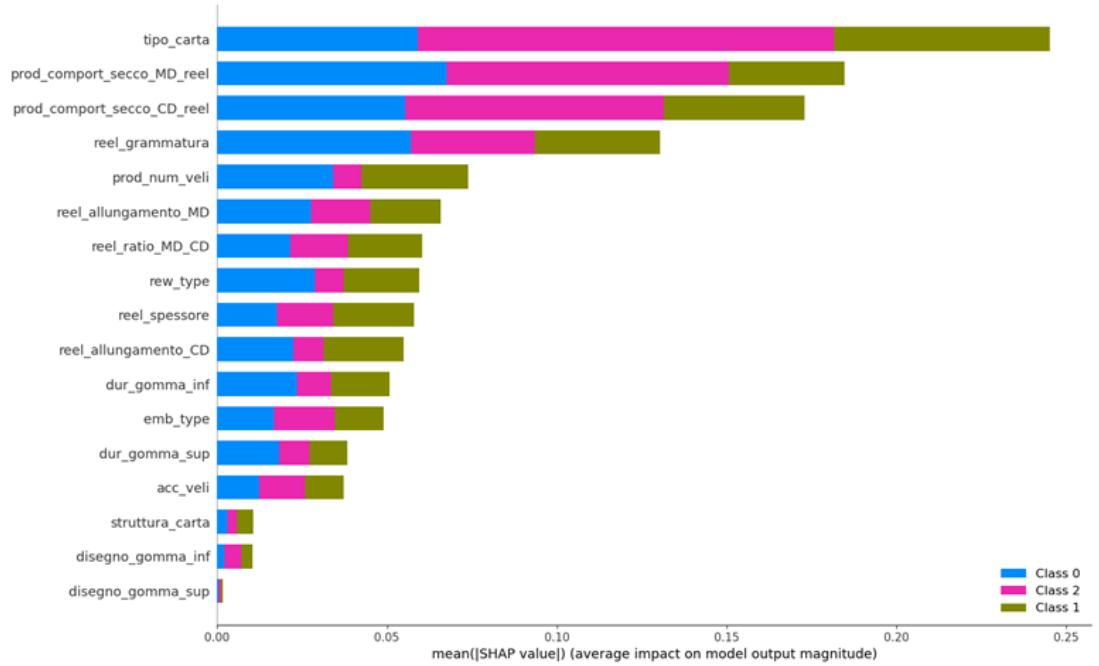

Figura 5.10: Importanza delle feature calcolata tramite i valori di Shapley. Accuracy score del modello: 0.845

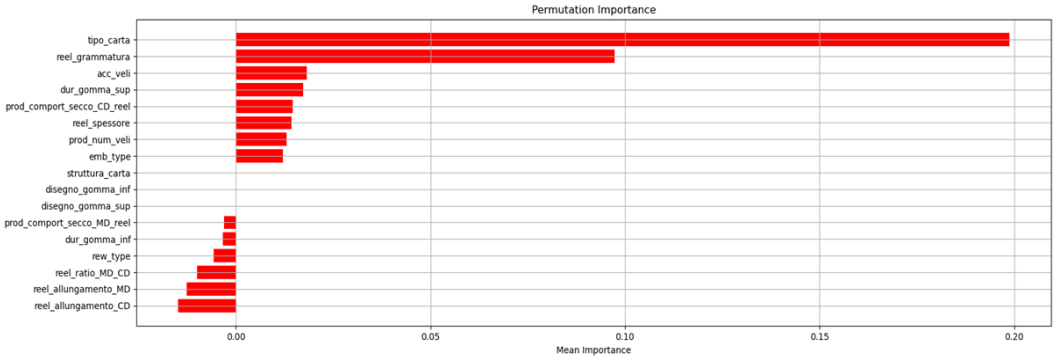

Figura 5.11: Importanza delle feature calcolata con Permutation Importance. Accuracy score del modello: 0.745

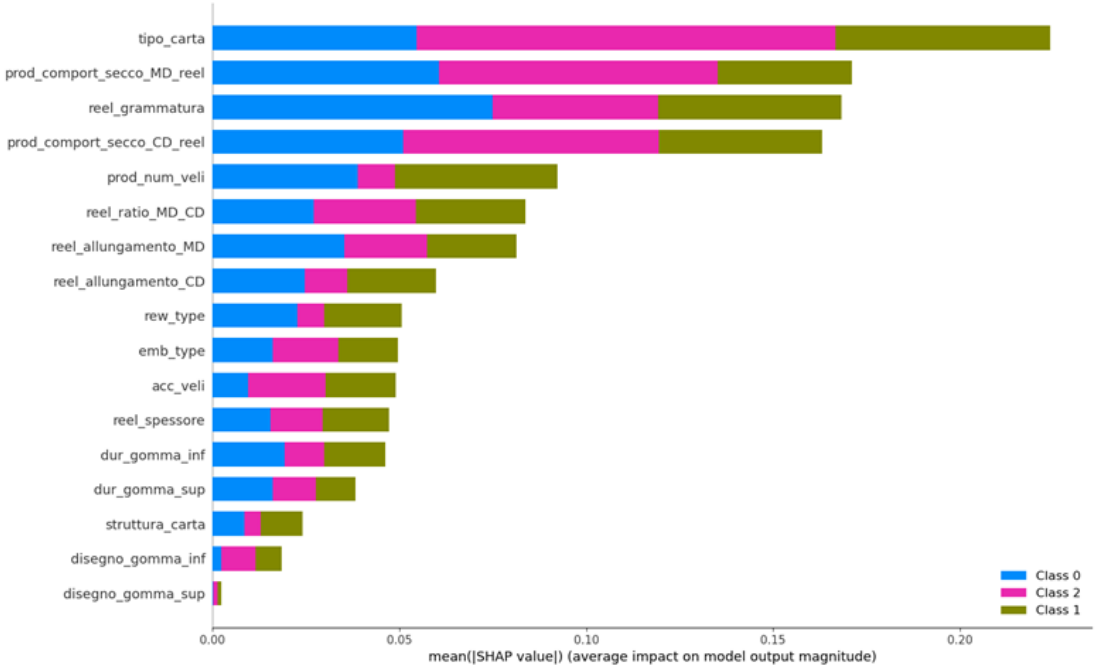

Figura 5.12: Importanza delle feature calcolata tramite i valori di Shapley. Accuracy score del modello: 0.745

marginale in tutto il dataset ma la media dei contributi marginali di ogni possibile combinazione di features.

La tabella 5.1 mostra il riassunto dei risultati di accuratezza al variare del modello e delle parametrizzazioni:

Parametrizzazione	**Modello**	
	MLP	Random Forest
Iperparametri standard	0.800	0.827
Iperparametri ottimizzati	0.863	0.882

Tabella 5.1: Accuracy score ottenuta in diversi casi

Capitolo 6

Conclusioni

Grazie ai modelli di Intelligenza Artificiale spiegabile è stato possibile seguire ad ogni passo le decisioni prese dagli algoritmi di machine learning, interpretarle ed estrapolare informazioni sui dati.

L'analisi è partita da un modello con un'accuratezza di 0.8, che, come dimostrato dal grafico, dipendeva da un maggiore numero di iterazioni dell'algoritmo. Il passo successivo è stato quello di confrontare i risultati di un modello addestrato con MLP e uno con *Random Forest* al variare degli iperparametri. In entrambi i casi si è avuto un incremento significativo nella precisione delle previsioni. Attraverso l'esecuzione di numerosi test è stato inoltre possibile ricavare alcune considerazioni: per entrambi i modelli, la presenza di un numero molto limitato di dati è un problema, poiché aumenta la probabilità che i set di allenamento e di testing siano molto diversi l'uno dall'altro, causando, di conseguenza, una maggiore variabilità nelle prestazioni del modello tra un test e il successivo. Un'altra osservazione riguarda nello specifico l'algoritmo *Random Forest*. Infatti, questo combina un insieme di alberi decisionali, ognuno addestrato su un sottoinsieme casuale dei dati, e questa aggregazione riduce la sensibilità dei singoli alberi alle variazioni degli iperparametri.

Successivamente ci si è concentrati sull'analisi delle features. In una prima fase sono stati utilizzati e messi a confronto due algoritmi, sia per valutare le loro prestazioni e capire quanto i loro risultati fossero simili sia per raccogliere informazioni sulle features. I due algoritmi, ignorando alcune differenze minime, restituiscono risultati analoghi. Questa somiglianza però viene meno nel momento in cui il modello non viene addestrato correttamente e la sua precisione nelle previsioni è bassa. L'alta variabilità dell'accuratezza del modello può essere spiegata nell'applicazione della funzione `train_test_split` a un dataset relativamente piccolo, che crea dei sottoinsiemi che possono differire di molto da un'iterazione all'altra. I due algoritmi vengono applicati ognuno a un sottoinsieme diverso: *Permutation Importance* si basa sui dati di testing mentre la *feature importance* calcolata col *Gini Index* utilizza i

dati di training, con l'effetto negativo di calcolare anche l'importanza (erroneamente alta) di caratteristiche non effettivamente utili alla predizione.

Infine, si è tentato di comprendere le motivazioni dietro i risultati molto variabili di *Permutation Importance*. Sono state infatti scoperte forti correlazioni presenti tra svariate feature del dataset, che hanno alterato la valutazione dell'importanza delle stesse. Inoltre, è stato eseguito un confronto con un ulteriore algoritmo di analisi delle *feature importances*. Quest'ultimo, basato sul calcolo dei valori di Shapley, si è scoperto essere molto più robusto alla correlazione tra le feature, restituendo risultati maggiormente coerenti tra loro.

In conclusione, l'uso dell'IA spiegabile nella mia ricerca ha permesso di ottenere una maggiore comprensione dei dati e dei processi analizzati. Il suo utilizzo ha anche evidenziato l'importanza di una raccolta e di un'analisi dei dati accurata e completa, al fine di ottenere risultati affidabili e utili per il miglioramento dei processi produttivi e la qualità dei prodotti. Nel caso in esame, attraverso l'implementazione di algoritmi di machine learning e l'analisi delle *feature importances*, è stata messa in evidenza, oltre alla necessità di raccogliere un numero maggiore di dati, l'elevata correlazione tra determinate feature. Per limitare questo problema e ricavare informazioni maggiormente significative, un possibile approccio futuro potrebbe comprendere un'attività di *feature selection*, per selezionare solo le feature più rilevanti.

Capitolo 7

Appendice A

In questa sezione verrà mostrato il codice utilizzato per implementare gli algoritmi e le funzionalità descritte nei capitoli precedenti.

7.1 Librerie

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler
import json
import pandas as pd
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from numpy import ravel
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
import shap
```

Listing 7.1: Librerie

7.2 Classe Dataset

```
class Dataset:
    def __init__(self):
        self.data = read_csv('data/BULK.CSV')
        self.data = self.data.drop('media_bulk', axis=1)
        self.features = self.data.columns[:-1]

    def split_data(self):
```

```
8           self.training_data, self.testing_data, self.training_labels,
9               self.testing_labels = train_test_split(
10              self.data.iloc[:, :-1], self.data.iloc[:, -1])
11
12      def encode_data(self):
13          encoder = OrdinalEncoder()
14          encoder.fit(self.training_data.iloc[:, 10:])
15          self.training_data.iloc[:, 10:] = encoder.transform(
16              self.training_data.iloc[:, 10:])
17          self.testing_data.iloc[:, 10:] = encoder.transform(
18              self.testing_data.iloc[:, 10:])
19
20      def save_data(self):
21          self.training_data.to_csv('data/trainingData.csv', index=False)
22          self.testing_data.to_csv('data/testingData.csv', index=False)
23          self.training_labels.to_csv('data/trainingLabels.csv', index=False)
24          self.testing_labels.to_csv('data/testingLabels.csv', index=False)
25
26      def recover_data(self):
27          self.training_data = read_csv('data/trainingData.csv')
28          self.testing_data = read_csv('data/testingData.csv')
29          self.training_labels = read_csv('data/trainingLabels.csv')
30          self.testing_labels = read_csv('data/testingLabels.csv')
31
32      def scale_data(self):
33          scaler = MinMaxScaler()
34          scaler.fit(self.training_data)
35          self.training_data = scaler.transform(self.training_data)
36          self.testing_data = scaler.transform(self.testing_data)
37
38      def save_params(self, params):
39          with open('config/modelConfiguration.json', 'w') as f:
40              json.dump(params, f)
41
42      def load_params(self):
43          with open('config/modelConfiguration.json', 'r') as f:
44              params = json.load(f)
45          return params
```

Listing 7.2: Classe Dataset

7.3 MLPClassifier

```
1 mlp = MLPClassifier(random_state=0)
2
3 parameters = {
4     'solver': ('adam', 'sgd', 'lbfgs'),
```

```
    'max_iter': (800, 1000, 2000),
    'learning_rate_init': (0.01, 0.001),
    'learning_rate': ('constant', 'invscaling', 'adaptive'),
    'hidden_layer_sizes': [(200, 100, 50, 25, 10), (50, 25),
        (100, 100, 50, 25)]
}

gs = GridSearchCV(mlp, parameters)
gs.fit(dataset.training_data, ravel(dataset.training_labels))

mlp = MLPClassifier(solver=gs.best_params_['solver'],
    max_iter=gs.best_params_['max_iter'],
    hidden_layer_sizes=gs.best_params_['hidden_layer_sizes'],
    learning_rate_init=gs.best_params_['learning_rate_init'],
    learning_rate=gs.best_params_['learning_rate'], random_state=0).fit(
    dataset.training_data, ravel(dataset.training_labels))
labels = mlp.predict(dataset.testing_data)
acc_score = accuracy_score(ravel(dataset.testing_labels), labels)
print('Accuracy score: ', acc_score)
```

Listing 7.3: MLPClassifier

7.4 RandomForestClassifier

```
rf = RandomForestClassifier(random_state=0)

params = {
    'n_estimators': [10, 200, 1000],
    'max_features': [None, 'sqrt', 'log2'],
    'min_samples_leaf': [1, 2, 3, 4, 5],
    'max_depth': [None, 10, 20]
}

grid_search = GridSearchCV(rf, params)
grid_search.fit(dataset.training_data, ravel(dataset.training_labels))

rf = RandomForestClassifier(n_estimators=grid_search.best_params_['
    n_estimators'], max_features=grid_search.best_params_['max_features'],
    min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
    max_depth=grid_search.best_params_['max_depth'], random_state=0)

rf.fit(dataset.training_data, ravel(dataset.training_labels))
predicted = rf.predict(dataset.testing_data)
print('Accuracy score: ',accuracy_score(dataset.testing_labels, predicted))
```

Listing 7.4: RandomForestClassifier

7.5 Permutation Importance

```
p = permutation_importance(rf, dataset.testing_data,dataset.testing_labels,
    n_repeats=50, random_state=0, scoring='accuracy', n_jobs=1)
dataset.testing_data = pd.DataFrame(dataset.testing_data)
dataset.testing_data.columns = dataset.features
sorted_idx = p.importances_mean.argsort()
for i in sorted_idx[::-1]:
    print(f"{dataset.testing_data.columns[i]:<8}:  "
          f"{p.importances_mean[i]:.3f}"
          f" +/- {p.importances_std[i]:.3f}")
```

Listing 7.5: Permutation Importance

7.6 SHAP

```
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(dataset.testing_data)
shap.summary_plot(shap_values, dataset.testing_data,
    feature_names=dataset.features)
```

Listing 7.6: SHAP

Bibliografia

[1] Che cos'è l'AI spiegabile? | IBM — ibm.com. https://www.ibm.com/it-it/topics/explainable-ai.

[2] Imran Ahmed, Gwanggil Jeon, and Francesco Piccialli. From artificial intelligence to explainable artificial intelligence in industry 4.0: A survey on what, how, and where. *IEEE Transactions on Industrial Informatics*, 18(8):5031–5042, 2022.

[3] Fernando Brügge. The rise of industrial AI and AIoT: 4 trends driving technology adoption — iot-analytics.com. https://iot-analytics.com/rise-of-industrial-ai-aiot-4-trends-driving-technology-adoption/.

[4] Abdul Rehman Javed, Waqas Ahmed, Thippa Gadekallu, Praveen Reddy, and Mamoun Alazab. A survey of explainable artificial intelligence for smart cities. *Electronics*, 12, 02 2023.

[5] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.