UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Hybrid Random Forests: un nuovo approccio alla classificazione delle immagini

Hybrid Random Forests: a new approach to image classification

Relatore

Prof. Domenico Ursino

Candidata

Sara Montese

ANNO ACCADEMICO 2021-2022

Sommario

Negli ultimi anni il numero di immagini digitali è cresciuto in modo estremamente rapido, introducendo nuove sfide nell'ambito dell'Image Classification che non erano emerse per dataset più ridotti. In particolare, la gestione di insiemi di dati in continuo aumento, in cui non solo la quantità di immagini di training, ma anche il numero di classi cresce nel tempo, costituisce un problema finora poco esplorato. In questa tesi si propone una variante della Random Forest standard, che combina nodi generativi e nodi discriminanti al fine di ovviare a tale problema. Questo nuovo approccio consente di integrare i dati di nuove classi in modo da estendere la foresta precedentemente costruita, piuttosto che addestrarla nuovamente da zero. Gli esperimenti svolti dimostrano che questo classificatore raggiunge risultati incoraggianti rispetto allo stato dell'arte, offrendo, in aggiunta, la possibilità di apprendere in modo incrementale.

Keyword: Machine Learning, Image Classification, Incremental Learning, Random Forest.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

L'ultimo decennio è stato segnato da incredibili progressi nel Machine Learning e dalla sua crescente presenza in campi sempre più vari. Anche se questo ramo dell'Intelligenza Artificiale ha dimostrato il suo valore su problemi difficili da risolvere fino ad ora, esso presenta un grande svantaggio: i modelli attuali non si adattano facilmente a dati su cui non sono stati precedentemente addestrati. In presenza di nuovi dati, si osserva il fenomeno della cosiddetta "*dimenticanza catastrofica*": il modello si concentra su queste nuove informazioni e "dimentica" quanto già processato. Inoltre, bisogna considerare che non è possibile memorizzare enormi quantità di dati, poiché ciò sarebbe oneroso a livello di risorse.

Con l'algoritmo trattato in questo lavoro di tesi si vuole avvicinare l'apprendimento automatico all'apprendimento umano, al fine di riprodurre la nostra capacità di imparare nel corso della vita, con pochi dati per volta. Il traguardo finale di questo progetto è introdurre un nuovo tipo di apprendimento per le Random Forest, detto *incremental learning*, che consente all'algoritmo di apprendere e adattarsi continuamente, ogni volta che emergono nuovi dati, anche dopo il training iniziale. In questo modo, si conservano le informazioni sulle classi di dati apprese, mentre si aggiorna il modello attuale per imparare quelle nuove. Le Random Forest sono ottimi candidati per questa nuova tipologia di apprendimento poiché, grazie alla loro struttura gerarchica, le modifiche possono essere apportate localmente, e quindi hanno un impatto solo su una frazione dei dati, con un costo computazionale ridotto. L'algoritmo è stato applicato nell'ambito dell'Image Classification, in quanto, negli ultimi anni, si è assistito ad una crescita esponenziale delle immagini digitali. La possibilità di condividere contenuti visivi attraverso i Social Media e l'avvento di fotocamere di alta qualità a prezzi accessibili, hanno reso le immagini onnipresenti nella nostra vita, introducendo nuove sfide per l'elaborazione di dataset con un elevato numero di classi.

In particolare, l'obiettivo di questo lavoro di tesi è stato ottimizzare i parametri dell'algoritmo, aggiungere nuove funzionalità e testarne le performance su set di dati complessi.

Al fine di illustrare al meglio il lavoro svolto, l'elaborato è stato strutturato come di seguito specificato:

- Nel Capitolo 1 viene presentata una panoramica sul Machine Learning, introducendo i concetti di base più importanti.
- Nel Capitolo 2 vengono analizzati il problema della classificazione e i principali algoritmi di Machine Learning utilizzati per tale scopo.
- Nel Capitolo 3 viene fornita una descrizione complessiva del progetto, illustrando in dettaglio come è stata organizzata la sua realizzazione.

- Nel Capitolo 4 vengono illustrati nel dettaglio il lavoro svolto e i risultati che il sistema prodotto ha permesso di raggiungere.
- Nel Capitolo 5 viene svolta un'analisi critica del progetto, individuando i punti di forza e i punti di debolezza che lo caratterizzano.
- Nel Capitolo 6 vengono tratte le conclusioni e delineati alcuni possibili sviluppi futuri.

CAPITOLO 1

Introduzione al Machine Learning

In questo capitolo verrà illustrato un quadro generale sul Machine Learning, introducendo le definizioni e i concetti di base più importanti. In particolare, nelle prime sezioni, si fornirà una panoramica sulle forme di apprendimento intelligente e sulle relative tecniche, con un approfondimento particolare per gli argomenti correlati al lavoro svolto in questa tesi. Successivamente, si mostrerà come costruire e validare un algoritmo di apprendimento automatico, per concludere, infine, con alcune delle numerose applicazioni di questo ramo dell'Intelligenza Artificiale.

1.1 Panoramica

Il Machine Learning, noto in italiano come *apprendimento automatico*, è la branca dell'Intelligenza Artificiale che fornisce ai computer la capacità di svolgere un'attività naturale per l'uomo, ovvero imparare e migliorare dall'esperienza. Con l'aumento dei Big Data, i database che immagazzinano dati sono enormi e in costante crescita, e la ricerca di informazioni rilevanti non può essere fatta manualmente. L'apprendimento automatico è diventato, dunque, fondamentale: se il sistema può imparare e adattarsi ai cambiamenti, il progettista del sistema non ha bisogno di prevedere e fornire soluzioni per tutte le possibili situazioni. Gli algoritmi di Machine Learning utilizzano, infatti, metodi matematico-computazionali per apprendere le informazioni direttamente dai dati, senza basarsi su un modello predeterminato.

1.1.1 Terminologia

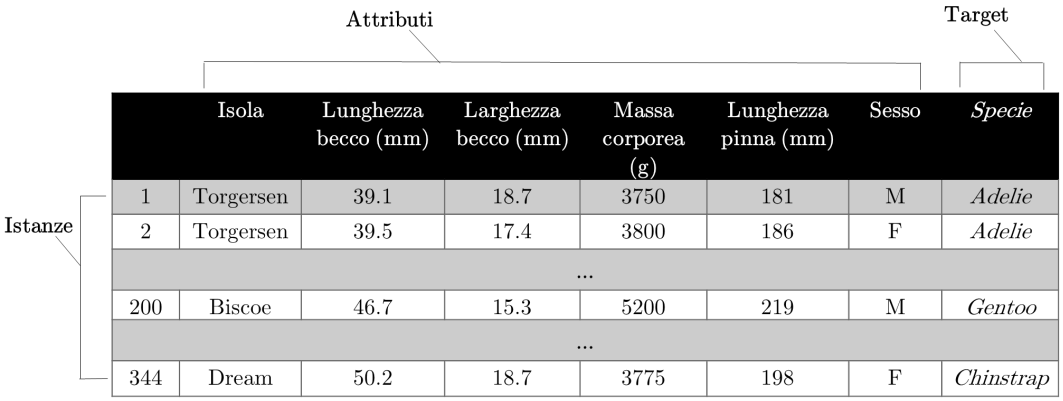
In questa sezione viene discussa la terminologia di base da conoscere prima di addentrarsi nel mondo dell'apprendimento automatico. Nello specifico, si prende come esempio il dataset PalmerPenguins[1], che consiste in 344 istanze di pinguini classificate in 3 diverse specie: Adelie, Gentoo e Chinstrap. Nella Figura 1.1 vengono riportati 3 termini chiave di un dataset:

- *Istanza*: detta anche "osservazione", rappresenta una riga del dataset; nel caso preso in esame, rappresenta uno dei 344 animali.
- *Attributo*: noto anche come "feature", è una proprietà dell'istanza utile per la sua caratterizzazione; nel caso in esame esempi di attributi sono la lunghezza e la larghezza del becco. Gli attributi sono il blocco di costruzione più importante di un set di dati e la loro selezione come input per l'algoritmo di apprendimento influisce direttamente

[1] PalmerPenguing dataset – `https://allisonhorst.github.io/palmerpenguins/`

sulla predizione del modello generato, quindi, al fine di generare un buon modello di predizione, vanno selezionate feature adeguate e di qualità.

- *Target*: detto anche "etichetta", è una caratteristica del set di dati di cui l'utente vuole prevedere il risultato futuro; nel caso preso in esame si vuole determinare la specie del singolo animale.



	Isola	Lunghezza becco (mm)	Larghezza becco (mm)	Massa corporea (g)	Lunghezza pinna (mm)	Sesso	*Specie*
1	Torgersen	39.1	18.7	3750	181	M	*Adelie*
2	Torgersen	39.5	17.4	3800	186	F	*Adelie*
...							
200	Biscoe	46.7	15.3	5200	219	M	*Gentoo*
...							
344	Dream	50.2	18.7	3775	198	F	*Chinstrap*

Figura 1.1: Termini chiave di un dataset

1.2 Tipologie di Apprendimento Automatico

Le due principali tecniche di Machine Learning sono l'apprendimento supervisionato e quello non supervisionato.

1.2.1 Unsupervised Learning

L'Apprendimento non Supervisionato (dall'inglese *Unsupervised Learning*) utilizza algoritmi di Machine Learning per analizzare e raggruppare serie di dati non etichettati, senza alcuna indicazione sull'output desiderato. Il fine di questa tipologia di apprendimento è individuare modelli e pattern nascosti tra i dati, senza il bisogno di un intervento umano.

I modelli di apprendimento non supervisionato sono utilizzati principalmente per due scopi: il raggruppamento (clustering) e la riduzione della dimensionalità (dimensionality reduction). Se il modello cerca di raggruppare dati simili sulla base delle loro caratteristiche si parla di un modello di *clustering* (Figura 1.2). Un esempio è rappresentato da un modello che segmenta i clienti di un'azienda in base ai loro profili. D'altra parte, se si ha un modello che trasforma i dati e li rappresenta con un numero minore di caratteristiche, parleremo di un modello di *riduzione delle dimensionalità*. Un esempio è rappresentato da un modello che riassume le molteplici caratteristiche tecniche delle automobili in pochi indicatori principali, come la cilindrata o la potenza massima.

Esempi classici di algoritmi di apprendimento non supervisionato sono il clustering k-means, i modelli a miscele gaussiane e l'analisi delle componenti principali (PCA).

1.2.2 Supervised Learning

Nell'Apprendimento Supervisionato (dall'inglese *Supervised Learning*), si vuole ottenere un modello per predire risultati o classificare dati sulla base delle loro caratteristiche. Tale tipologia di apprendimento utilizza un dataset di training per insegnare ai modelli a produrre

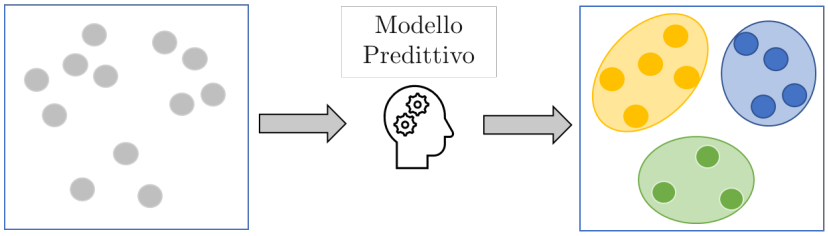


Figura 1.2: Algoritmo di Apprendimento non Supervisionato

l'output desiderato. Questo set di dati include input e output corretti, che permettono al modello di imparare nel tempo. Ad esempio, si potrebbe addestrare un algoritmo supervisionato utilizzando una serie di immagini di animali comuni e le loro etichette corrispondenti (come tartaruga, gatto e gallina). L'algoritmo sfrutterà caratteristiche utili ricavate dalle immagini, come il numero di gambe o il colore, per trovare modelli che colleghino le immagini con le rispettive etichette. Dopo l'addestramento, è possibile utilizzare l'algoritmo completamente addestrato per tentare di prevedere le etichette di una nuova serie di immagini sconosciute (Figura 1.3). Tali etichette possono essere numeri o categorie.

Si parla di modello di *regressione* quando si vuole predire un'etichetta di tipo numerico. Per esempio, è possibile costruire un modello che predica la temperatura di un dato giorno, o il prezzo di una casa. D'altra parte, è possibile definire un modello che predica una categoria, come una mail "spam" e "non spam", sulla base di caratteristiche date. In tal caso si parla di modello di *classificazione*.

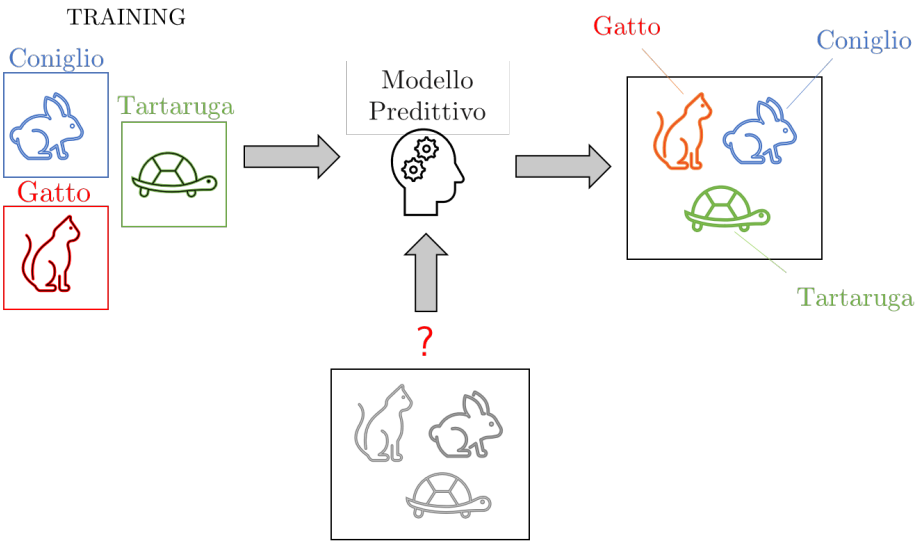


Figura 1.3: Algoritmo di Apprendimento Supervisionato

L'Apprendimento Automatico Supervisionato è il tipo più comune usato oggi ed esempi classici di algoritmi sono la regressione lineare, le foreste casuali, le macchine vettoriali di supporto (SVM) e le reti neurali.

Nella Figura 1.4 vengono riportati i diversi tipi di output attesi, in base alle diverse tipologie di apprendimento automatico:

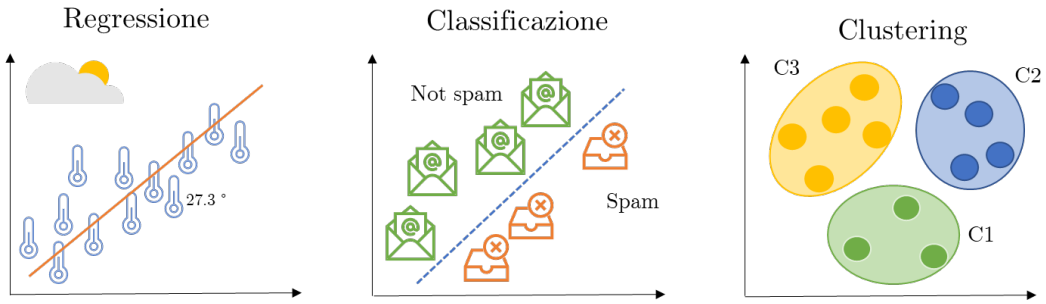


Figura 1.4: Diversi tipi di output attesi da un algorimo di ML

1.3 Processo di Apprendimento Automatico

1.3.1 Costruzione del Modello

Ci sono quattro step per costruire e utilizzare un modello di apprendimento automatico [9]:

1. *Pre-elaborazione*. Si acquisiscono e si preparano i dati che andranno utilizzati. Comprende i seguenti compiti:
   - estrazione delle caratteristiche dai dati grezzi;
   - pulizia e formattazione dei dati;
   - rimozione delle feature superflue e/o di quelle altamente correlate;
   - normalizzazione delle feature (Feature Scaling);
   - divisione del dataset in training set e test set.
2. *Apprendimento* o *addestramento*. Viene selezionato l'algoritmo ottimale per eseguire il compito desiderato, facendo riferimento alle metriche utilizzate per misurare le prestazioni di un modello di apprendimento automatico (Sezione 1.3.3). In questa fase, inoltre, è fondamentale definire accuratamente il set di parametri di configurazione dell'algoritmo (*iperparametri*).
3. *Valutazione*. Consiste nel misurare quanto accuratamente un algoritmo è in grado di predire dei risultati per dati mai visti prima. Per fare ciò, una volta che il modello è stato addestrato sul training set, esso viene testato su nuovi dati.
4. *Previsione*. Dopo che il modello è stato messo a punto e valutato, il passo successivo è quello di usarlo per fare previsioni. Questo viene effettuato utilizzando il set di dati di test.

1.3.2 Training, Validation e Test set

Per sapere come un modello si adatterà ai nuovi dati è necessario provarlo sulle istanze stesse. A tale scopo il dataset fornito all'algoritmo viene diviso in due parti:

- *Training*. Questo viene suddiviso, a sua volta, in:
  - *Training set*: campione di dati utilizzato per addestrare il modello e tarare gli iperparametri.
  - *Validation set*: campione di dati utilizzato per convalidare le prestazioni del modello durante l'addestramento, fornendo informazioni utili per perfezionare le configurazioni del modello.

- *Testing*. Consiste nel campione di dati su cui valutare le prestazioni finali dell'algoritmo.

Nella Figura 1.5 viene illustrato il processo di costruzione e convalida di un modello di apprendimento automatico.

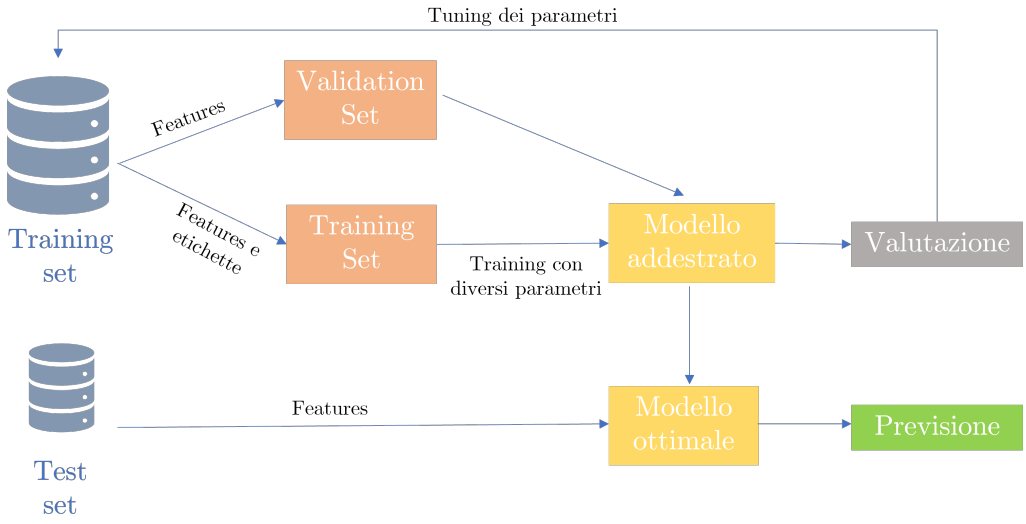


Figura 1.5: Processo di Training e Testing di un modello di ML

Per evitare di utilizzare troppi dati di training nel validation set e non averne a sufficienza per addestrare l'algoritmo, una tecnica comune è quella della *validazione incrociata* (o cross-validation). Tale procedura prevede che il campione di training venga suddiviso in partizioni (folds) di uguale dimensione; successivamente esso viene allenato su una parte di questi e convalidato sulla rimanente. L'operazione è ripetuta per le diverse combinazioni dei sottocampioni ed il risultato finale è costituito dalla media delle performance delle varie iterazioni. In aggiunta, questa tecnica è utile anche per limitare gli effetti di dati incompleti o non accurati, e per ridurre il problema del cosidetto "overfitting" (Sezione 1.3.4).

Iterazione 1	Test	Train	Train	Train	Train
Iterazione 2	Train	Test	Train	Train	Train
Iterazione 3	Train	Train	Test	Train	Train
Iterazione 4	Train	Train	Train	Test	Train
Iterazione 5	Train	Train	Train	Train	Test

Figura 1.6: Esempio di 5-folds cross validation

1.3.3 Valutazione delle Prestazioni

Il corretto funzionamento degli algoritmi di apprendimento automatico dipende dai dati in ingresso. Se si forniscono pochi dati l'algoritmo potrebbe non avere abbastanza esperienza e commettere molti errori; d'altra parte, un numero troppo elevato di dati potrebbe comportare una scarsa capacità di generalizzazione del modello.

La valutazione dei metodi di classificazione e predizione può essere fatta secondo i seguenti criteri:

- *Accuratezza*: è la capacità del modello di fare previsioni corrette su dati nuovi o precedentemente non visti.
- *Velocità*: è il tempo di esecuzione computazionale che include la generazione e l'utilizzo del modello di apprendimento.
- *Robustezza*: è la capacità del modello di effettuare previsioni corrette anche in presenza di dati mancanti o errati.
- *Scalabilità*: è l'abilità del modello di essere efficiente sia per piccoli set di dati, che per database di grandi dimensioni.

1.3.4 Underfitting e Overfitting

Underfitting e Overfitting sono due delle cause più comuni di scarsa accuratezza di un modello di apprendimento automatico.

Si parla di *Underfitting* quando un modello presenta alti errori di predizione sia per i dati di training che per quelli di test. In questo caso, il modello non riesce a cogliere in modo significativo la relazione tra i valori di input e le variabili target. Ciò si verifica quando esso è troppo semplice, ovvero quando le caratteristiche di input non sono abbastanza esplicative per descrivere bene l'obiettivo. Le previsioni, in questo caso, soffrono di un'eccessiva discrepanza (*bias elevato*).

Si parla di *Overfitting* quando un modello ha memorizzato eccessivamente i dettagli nei dati di training e non è in grado di generalizzare l'apprendimento ad un nuovo set di dati. Questo è il motivo per cui un modello sovra-adattato comporta un'accuratezza di test molto scarsa. In questi casi, la varianza delle previsioni diventa elevata perché il modello è troppo sensibile ai dati addestramento (*varianza elevata*).

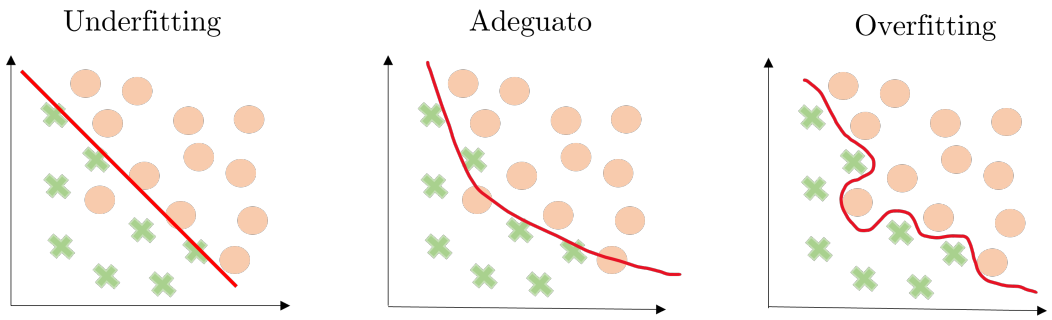


Figura 1.7: Esempio di Underfitting e Overfitting

Dunque, è necessario trovare il giusto compromesso tra un modello troppo semplice ed uno con un eccessivo numero di parametri.

1.4 Applicazioni

Il Machine Learning ha una vasta gamma di utilizzi nelle aziende di oggi, nel settore bancario, nell'e-commerce, nella medicina, nella scienza, etc., e tale varietà è destinata ad aumentare nel tempo.

L'*image recognition* è tra le applicazioni più comuni: sempre più numerosi sono gli algoritmi per l'identificazione di volti, persone, oggetti, luoghi e testi.

Sistemi di *riconoscimento vocale* come Alexa, Siri e Google Assistant stanno cambiando il modo con cui le persone interagiscono con gli smartphone, le macchine e le case.

I suggerimenti di Netflix e YouTube, le informazioni che appaiono sul feed di Facebook e i prodotti consigliati su Amazon, ad esempio, sono *algoritmi di raccomandazione* alimentati dall'apprendimento automatico.

Le *auto a guida autonoma* fanno uso di forme di apprendimento intelligente per identificare ciò che le circonda durante la guida (come persone, semafori, cartelli, e così via).

Questi sistemi hanno rivoluzionato la *diagnosi medica*, grazie alla capacità degli algoritmi di riconoscere pattern che vanno oltre la percezione umana, in base a sintomi e immagini mediche del paziente.

Nell'ambito della *sicurezza informatica* esistono sistemi per il filtraggio dello spam e del malware via e-mail, o per il rilevamento delle frodi online. L'apprendimento automatico è ampiamente utilizzato nel trading del *mercato azionario*, così come nelle *helpline automatiche*, dove gli utenti non interagiscono con operatori umani, bensì con delle macchine.

In conclusione, negli ultimi anni, sono stati fatti passi da gigante per quanto riguarda le forme di apprendimento intelligente. Le possibilità di sviluppo futuro di questo ramo dell'Intelligenza Artificiale sono ancora molte, anche nei settori più impensabili, come la giustizia.

CAPITOLO 2

Algoritmi di Classificazione

Questo capitolo sarà dedicato al problema della classificazione e agli algoritmi di Machine Learning comunemente utilizzati per tale scopo. Verranno introdotti gli alberi decisionali come meccanismo per effettuare classificazioni e verrà descritto il processo di costruzione e il funzionamento del corrispondente algoritmo di apprendimento. La sezione finale del capitolo approfondirà nel dettaglio il classificatore utilizzato nel presente lavoro di tesi, ovvero, le Random Forest.

2.1 Classificazione

La classificazione è definita come il processo di riconoscimento, comprensione e raggruppamento di oggetti e idee in categorie prestabilite [10]. Dal riconoscimento di determinate caratteristiche è possibile classificare un oggetto sconosciuto attribuendo ad esso una classe di appartenenza, definita a priori.

In questo lavoro di tesi, il modello sarà addestrato al fine della classificazione supervisionata. L'obiettivo è quello di utilizzare i dati a disposizione, dei quali sono note le classi di appartenenza, per costruire un classificatore in grado di apprendere dai dati storici in ingresso e riuscire ad assegnare un'etichetta ad un dato mai visto prima.

2.1.1 Approccio discriminante vs generativo

Generativo e *discriminante* sono termini che designano due diversi metodi di classificazione nell'apprendimento automatico, ovvero mezzi con cui viene assegnata una classe ad un datapoint.

La classificazione *generativa* viene effettuata creando un modello per ciascuna delle classi che si cerca di rilevare. L'algoritmo cercherà di definire uno o più criteri che permettano di identificare e descrivere i dati forniti in ingresso. Se, per esempio, si prova a differenziare un cane da un gatto in base al peso e all'altezza, un algoritmo generativo cercherà di imparare cosa caratterizza un cane o un gatto, e poi utilizzerà questi criteri per classificare nuove istanze (Figura 2.1). Proiettando ciascun datapoint su un piano in base a tali criteri, si crea una zona per ogni classe, e tutti i punti situati all'interno di questa zona verranno identificati come appartenenti a tale categoria.

La classificazione *discriminante* viene effettuata tracciando un confine tra i punti, in modo da isolare gruppi di datapoint e, dunque, classi. A differenza del metodo generativo, questo tipo di classificazione delimita semplicemente un confine senza apprendere un modello di

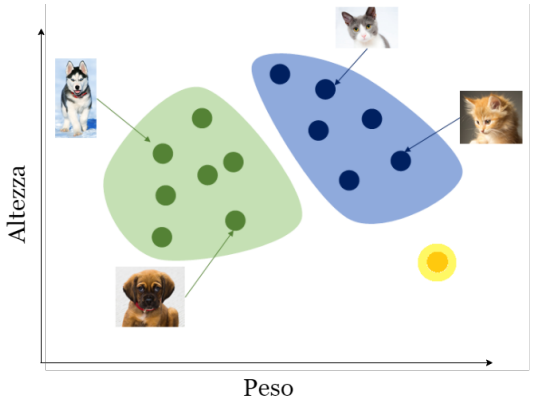


Figura 2.1: Esempio di classificazione generativa

classe. Infatti, se si considera l'esempio precedente, un algoritmo discriminante traccerà un confine tra "cane" e "gatto", senza però imparare o sapere cosa fa l'uno o l'altro (Figura 2.2).

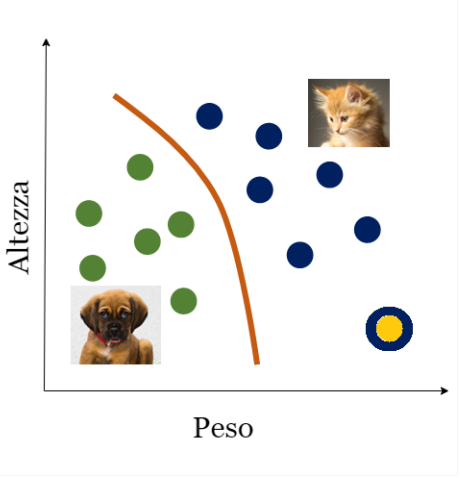


Figura 2.2: Esempio di classificazione discriminante

Se la classificazione discriminante risulta spesso più facile da realizzare, anche il metodo generativo ha i suoi vantaggi: uno dei più noti è la capacità di riconoscere una nuova classe. Infatti, poiché l'algoritmo ha costruito un modello delle classi su cui è stato addestrato, se si trovasse di fronte ad un'osservazione che non corrisponda a nessun modello, sarebbe quindi possibile assegnare a tale dato una nuova classe, o almeno contrassegnarlo come un outlier (nella Figura 2.1, questa situazione è rappresentata dal punto al di fuori delle due aree che contrassegnano le classi "cane" e "gatto", appartenente ad una nuova classe in giallo). Di fronte alla stessa situazione, un algoritmo discriminante classificherà semplicemente il nuovo datapoint in base a quale lato del confine si trovi (nella Figura 2.2, il nuovo dato è segnato come appartenente alla classe "gatto"). Un algoritmo generativo ha, quindi, la capacità di rilevare elementi appartenenti a potenziali nuove categorie di dati che non erano presenti durante la fase di training. Questa proprietà è molto vantaggiosa oggi, soprattutto nel contesto dell'apprendimento incrementale, un tipo di apprendimento in cui l'algoritmo viene addestrato su un flusso di dati che si evolve nel tempo.

2.2 Alberi decisionali

Il *Decision Tree* (o albero decisionale) è uno degli algoritmi supervisionati di classificazione e regressione più comuni in ambito di Machine Learning. Si tratta di un sistema con n variabili in input (le feature), derivate dall'osservazione dell'ambiente, ed m variabili in output, che identificano la decisione da intraprendere [6]. Gli alberi decisionali utilizzano un approccio top-down; cominciando dal nodo radice, che contiene l'intero dataset, l'algoritmo sceglie la feature più accurata per ottimizzare la previsione della variabile target scelta; i dati vengono, quindi, divisi in rami diversi a seconda delle specifiche richieste sulla feature stessa. L'algoritmo effettua lo stesso procedimento ad ogni nodo fino a quando si raggiunge la profondità prefissata, o fino a quando non ci sono più variabili per distinguere ulteriormente i dati. La decisione finale si trova nei nodi terminali, detti *foglie*, localizzati più in basso. Un tale approccio di modellazione permette all'attributo con maggiori capacità predittive di essere collocato al livello superiore come radice, e agli attributi meno significativi di essere collocati ai livelli inferiori come foglie. Gli alberi decisionali usati per predire variabili di tipo categorico sono chiamati *alberi di classificazione*; se usati per predire dati numerici, sono detti di *regressione*. Il risultato finale è un albero composto da:

- *Nodi decisionali*: nodi che applicano una condizione su una particolare feature e che presentano due o più diramazioni.
- *Archi decisionali*: rappresentano possibili valori della feature considerata.
- *Foglie*: contengono la decisione di classificazione o previsione della variabile target in considerazione.

Il modello di previsione nell'albero è rappresentato dal cammino (*path*) dalla radice alle foglie. Nella Figura 2.3 viene mostrato un esempio di albero decisionale binario (in cui ad ogni nodo corrispondono due soli rami) per scegliere se accettare o meno un'offerta di lavoro.

2.2.1 Costruzione di un albero

La costruzione di un albero binario di classificazione richiede tre fasi distinte:

1. *Crescita dell'albero*: consiste nella scelta del criterio di splitting di ogni nodo nei nodi figli.
2. *Potatura dell'albero*: consiste nella definizione di un criterio d'arresto nella costruzione dell'albero.
3. *Assegnazione delle classi*: consiste nell'individuazione di una regola per l'attribuzione di una delle classi ai nodi terminali.

La costruzione dell'albero segue una procedura ricorsiva; in linea di principio, l'algoritmo continua il partizionamento dei nodi fino a quando tutte le foglie contengono osservazioni di una singola classe. Nel seguito viene illustrata brevemente ciascuna delle tre fasi.

La prima fase (1) consiste nella scelta del criterio di splitting di ciascun nodo nei nodi figli. Tra le possibili feature, è necessario scegliere quella che garantisca la suddivisione delle osservazioni presenti nel nodo in sottogruppi il più possibile omogenei al loro interno, ed eterogenei tra loro. Per selezionare la partizione migliore, fra tutte quelle possibili, la decisione in ogni nodo viene presa in base ad una metrica chiamata *purezza*. Un nodo è puro al 100% quando tutti i suoi dati appartengono ad una sola classe. Nella maggior parte dei casi, i nodi terminali non sono puri, ma contengono osservazioni di più classi, come mostrato

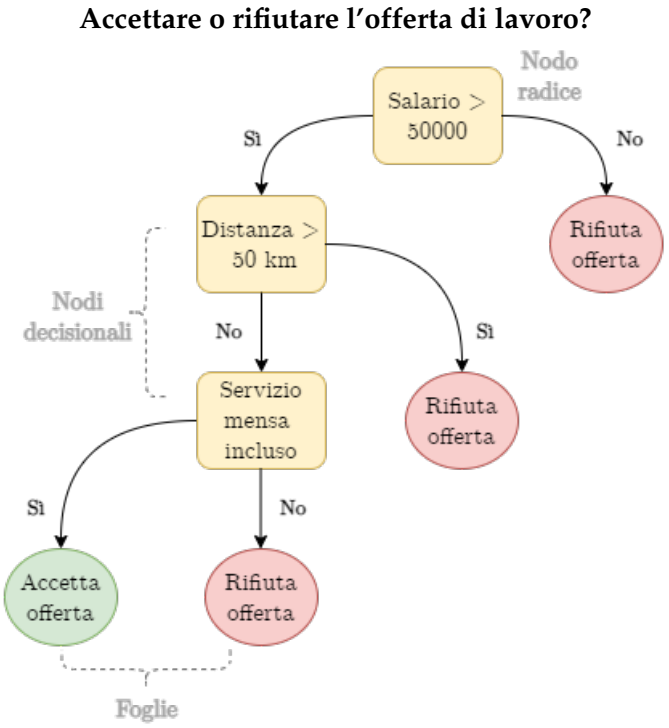


Figura 2.3: Esempio di Decision Tree

nella Figura 2.4.

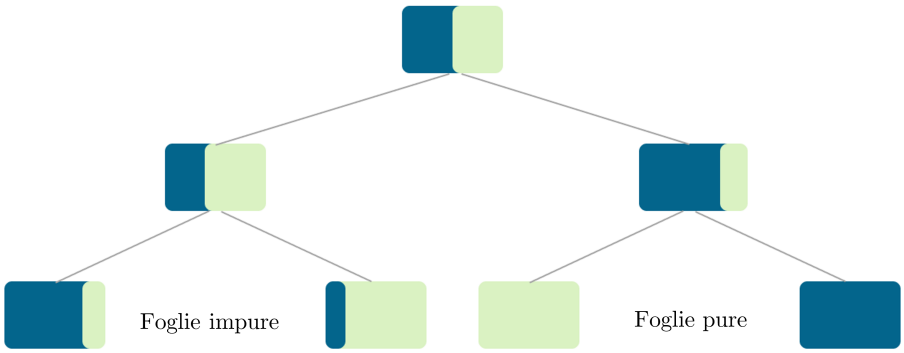


Figura 2.4: Esempio di albero con foglie di diversa purezza [1]

Per ottimizzare il modello è necessario raggiungere la massima purezza ed evitare l'impurità. Per fare ciò, l'indice statistico di riferimento è *Gini*; esso misura quanto spesso un elemento scelto a caso dal dataset verrebbe etichettato in modo errato se gli venisse assegnata casualmente una classe [11]. Considerando un dataset che contiene osservazioni di N classi, l'indice è definito come segue:

$$Gini = \sum_{k=1}^{N} p_k(1 - p_k) = 1 - \sum_{k=1}^{N} p_k^2$$

con p_k la probabilità di un oggetto di appartenere alla classe $k \in 1, 2, \ldots, N$. Con questo calcolo è possibile misurare l'impurità del singolo split; la feature che produce l'impurità più bassa si rivela essere la migliore per dividere il nodo corrente. Tale procedimento continua per ogni nodo successivo, usando le feature rimanenti.

L'altra metrica utilizzata è l'*Entropia*, che può essere vista come una misura del disordine o, in altre parole, del grado di casualità di un gruppo di osservazioni. Essa va da un valore minimo pari a 0, che indica un insieme di dati completamente omogeneo, al valore 1, che denota il massimo disordine, inteso come la possibilità di trovare nella stessa classe quanti più dati con parametri diversi. Considerando un dataset che contiene osservazioni di N classi, l'indice è definito come segue:

$$Entropia = -\sum_{k=1}^{N} p_k \log p_k$$

con p_k la probabilità di un oggetto di appartenere alla classe $k \in 1, 2, \ldots, N$. Tale metrica è più pesante dal punto di vista computazionale a causa della presenza del logaritmo nell'equazione. È importante sottolineare che minimizzare l'Entropia equivale a massimizzare il guadagno di informazione (*information gain*), ovvero la misura delle informazioni che una feature fornisce su una particolare classe.

La seconda fase (2) consiste nella definizione di un criterio d'arresto alla costruzione dell'albero stesso. Infatti, per evitare l'overfitting, migliorare la struttura dell'albero e avere un algoritmo computazionalmente fattibile, un ulteriore passo è la semplificazione dell'albero effettuata rimuovendo alcuni dei rami creati (la cosiddetta "potatura" o *"pruning"*). Poiché la costruzione di un albero è una procedura ricorsiva, è necessaria la definizione di una o più regole di arresto, al verificarsi delle quali il processo termina. Le proprietà desiderabili di una regola di arresto sono la semplicità e il potere discriminatorio. In base alla prima proprietà, fra più regole di stop si sceglie quella che determina l'albero di dimensione minore, e quindi più facilmente leggibile in fase di interpretazione dei risultati. La seconda proprietà riguarda, invece, l'esigenza di ottenere alberi in grado di distinguere nel modo più efficace possibile osservazioni appartenenti a classi diverse. Le due proprietà sono evidentemente contrapposte e difficilmente conciliabili. Solitamente, le regole di arresto nella costruzione di un albero sono basate sul numero minimo di nodi finali, o sulla profondità massima dell'albero.

Una volta che la struttura ad albero è stata potata, lo step finale (3) prevede l'assegnazione di un'etichetta di classe ad ogni nodo terminale. Si distinguono tre diversi casi:

a) La foglia comprende osservazioni appartenenti ad una sola classe: ad essa viene assegnata la label corrispondente alle unità che ne fanno parte.

b) La foglia comprende dati di classi diverse, ma una di queste ha frequenza superiore alle altre: la classe della foglia sarà quella con frequenza massima.

c) Le osservazioni della foglia appartengono a classi diverse con stessa frequenza; in questo caso, si ricorre ad un'assegnazione casuale, salvo intervento del programmatore, che ne può effettuare una diversa.

2.2.2 Vantaggi e Svantaggi

In questa sezione vengono analizzati i punti di forza e le limitazioni di un albero decisionale. Per ciò che concerne i vantaggi, abbiamo che:

- il modello è semplice e poco costoso da costruire;
- può trattare feature irrilevanti, eseguendo implicitamente la selezione delle variabili più importanti;
- può gestire sia dati numerici che categorici;
- è un processo di classificazione rapido e trasparente;

- la visualizzazione e l'interpretazione dei risultati sono facili.

Per ciò che concerne gli svantaggi, invece, abbiamo:

- *Overfitting*: spesso gli alberi decisionali si adattano eccessivamente ai dati di training, e non sono poi in grado di generalizzare quanto appreso; ciò può essere risolto imponendo dei vincoli sui parametri del modello (Sezione 2.2.1, fase 2).
- *Instabilità*: piccole variazioni nei dati di training potrebbero generare alberi completamente diversi.
- *Algoritmo poco adatto per problemi complessi*: un numero elevato di dati costringe un albero a produrre numerosi nodi, comportando lunghi tempi di addestramento e un'eccessiva complessità del modello.

2.2.3 Ensemble Learning

L'*Ensemble Learning* è una tipologia di apprendimento automatico in cui più modelli, detti *weak learners*, sono addestrati per risolvere un problema e combinati per ottenere risultati migliori. L'ipotesi principale è che da modelli deboli, se combinati correttamente, si possono ottenere modelli più accurati e robusti. Il risultato finale è dato, nel caso di problemi di classificazione, dalla previsione di maggioranza dei vari modelli; in caso di problemi di regressione, il risultato è ottenuto dalla media delle previsioni.

È possibile classificare gli algoritmi di Ensemble Learning in base al metodo di combinazione dei modelli deboli; boosting e bagging ne sono due esempi.

L'idea centrale del **boosting** è l'implementazione di algoritmi omogenei in modo sequenziale, dove ognuno di essi cerca di migliorare la stabilità del modello concentrandosi sugli errori effettuati dall'algoritmo precedente. Dopo aver terminato la fase di apprendimento, il risultato finale si ottiene da una combinazione ponderata delle previsioni dei vari modelli.

Nel processo di **bagging** (*Bootstrap Aggregating)*, prima si estraggono n campioni dal training set, poi questi sottoinsiemi vengono usati per addestrare n modelli deboli dello stesso tipo. Tale sottocampionamento, detto *bootstrapping*, avviene in modo casuale e con ripetizione, ovvero considerando più volte gli stessi dati; ciò consente di produrre un modello finale con una varianza ridotta. Per effettuare una previsione, ognuno degli n modelli viene alimentato con un sottocampione di test. La previsione finale è data dalla media delle previsioni di ciascun learner (in caso di regressione) o dalla maggioranza delle previsioni (in caso di classificazione). Nella Figura 2.5 viene illustrata una panoramica del processo di bagging. Uno dei vantaggi chiave di questo metodo è che può essere eseguito in parallelo poiché non c'è dipendenza tra gli stimatori. Applicando tale procedimento ad un gruppo di alberi decisionali, si ottiene uno dei più potenti algoritmi di Machine Learning, nonché il classificatore utilizzato nel presente lavoro di tesi, ovvero le Random Forest.

2.3 Random Forest

Le Random Forest sono un algoritmo di apprendimento automatico ottenuto dall'aggregazione, tramite bagging, di più alberi decisionali. Pòssono essere utilizzate sia per problemi di regressione che di classificazione; questi ultimi sono quelli utilizzati nel nostro caso.

La foresta, oltre all'assegnazione di sottocampioni del training set ai singoli stimatori, aggiunge un ulteriore grado di casualità: il *feature bagging*. In un normale albero decisionale, quando è il momento di dividere un nodo, si considera ogni possibile feature e si sceglie quella che produce la maggior separazione tra le osservazioni nel nodo di sinistra e quelle nel

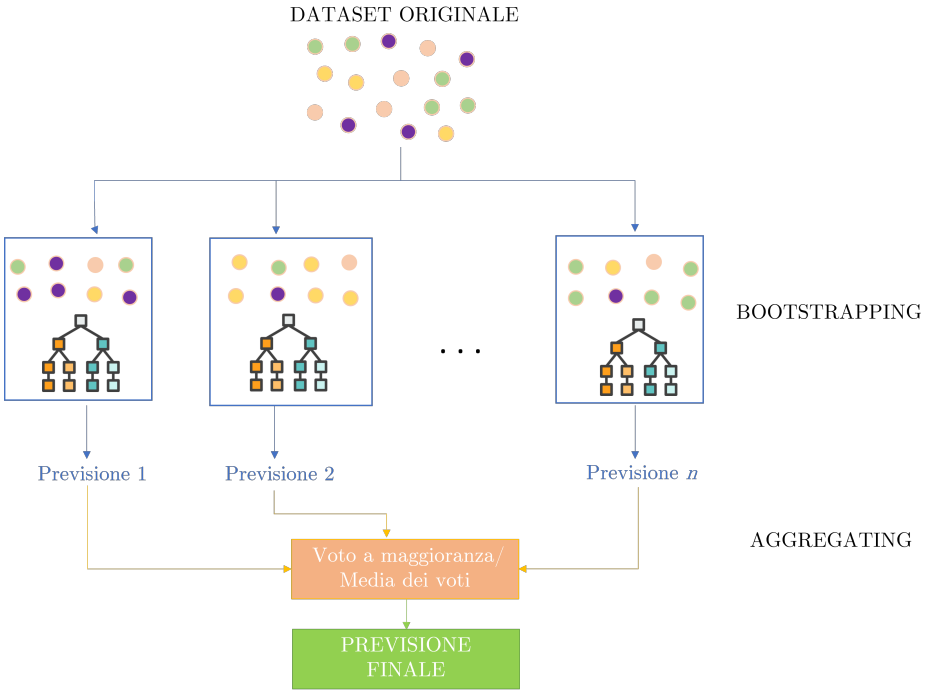


Figura 2.5: Esempio di bootstrap aggregating

nodo di destra. Al contrario, ogni albero in una Random Forest ricerca la feature migliore in un sottoinsieme casuale di tutte le feature. In questo modo, la foresta è costituita da alberi che non solo sono addestrati su diversi set di dati, ma usano anche variabili diverse per prendere decisioni. Ogni singolo albero nella foresta produce una previsione di classe, e la classe con più voti diventa poi la predizione finale del modello.

Nella Figura 2.6 viene illustrato un esempio di feature bagging. L'albero decisionale tradizionale può scegliere tra tutte e quattro le feature per decidere come dividere il nodo; si suppone che esso scelga di utilizzare la feature 1 (sottolineata), in quanto essa divide i dati in gruppi il più possibile eterogenei. Spostando l'attenzione sulla Random Forest, si

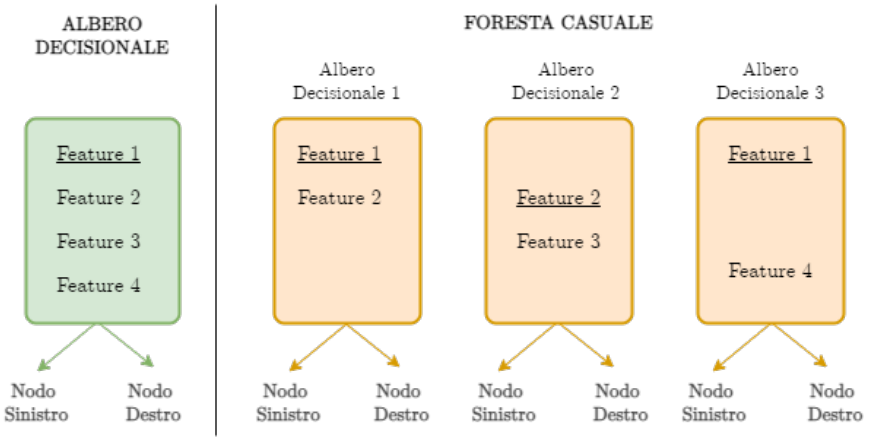


Figura 2.6: Esempio di feature bagging

nota, ad esempio, che il secondo albero decisionale di cui essa si compone considera solo un sottoinsieme casuale di variabili per la sua decisione di splitting dei nodi. Poiché tale albero non può vedere la feature 1, esso è costretto ad eleggerne un'altra (nell'esempio, la variabile scelta è la 2).

La maggiore diversità e minore correlazione dei singoli alberi decisionali portano ad una previsione nell'insieme più accurata di quella di ogni singolo albero.

Per fare il collegamento con i concetti di approccio generativo e discriminante, spiegati nella Sezione 2.1.1, gli algoritmi di Random Forest sono stati, fino ad ora, solo algoritmi discriminanti. Cioè, ogni nodo della foresta esegue una classificazione discriminante su uno o più criteri per ripartire progressivamente i dati. L'algoritmo discusso in questo lavoro di tesi introduce nodi che utilizzano una classificazione generativa, e li combina con nodi discriminanti, dando origine ad una Random Forest ibrida.

2.3.1 Gli iperparametri

In generale, un algoritmo di apprendimento automatico presenta un insieme di iperparametri che consentono ad esso di adattarsi a set di dati specifici. Quando esso viene implementato, di solito, viene messo a disposizione un set predefinito di iperparametri; è tuttavia consigliabile non utilizzare i parametri di default, ma personalizzare tali valori in base al contesto.

L'operazione dedicata alla scelta dei parametri ottimali per un modello di Machine Learning è nota come *tuning degli iperparametri*. È necessario, dunque, comprendere a pieno tali variabili, poiché esse sono fondamentali per accrescere la potenza predittiva del modello o per realizzarne uno più veloce nelle prestazioni.

Gli iperparametri del modello approfondito in questa tesi, i quali controllano lo sviluppo della struttura della Random Forest e di ogni singolo Decision Tree, sono stati scelti riferendosi allo stato dell'arte attuale [7]. Tra i principali iperparametri di una Random Forest, vi sono:

- *n_estimators*: rappresenta il numero di alberi nella foresta. Scegliere un numero di stimatori elevato può aiutare ad ottenere performance migliori; tuttavia, ciò è vero fino ad una certa soglia, dopodiché l'accuratezza del modello si attesta su un valore costante. Inoltre, un elevato numero di alberi comporta certamente una maggiore complessità computazionale del modello.

- *min_samples_leaf*: rappresenta il numero minimo di campioni che un nodo foglia deve contenere dopo essere stato diviso. Quando il valore del parametro è molto basso, la foresta è sovradattata ai dati di training; aumentando tale valore, le performance del modello migliorano rapidamente. D'altra parte, il parametro non dovrebbe essere troppo elevato, altrimenti si rischia di incorrere nell'underfitting.

- *max_features*: rappresenta il numero massimo di feature da selezionare casualmente per scegliere il miglior split. Inizialmente, le prestazioni del modello migliorano all'aumentare di tale valore. Tuttavia, superata una certa soglia, la diversità di ogni singolo albero si riduce, perdendo di vista il vero senso della foresta.

- *random_state*: controlla la casualità del bootstrapping dei campioni usati nella costruzione degli alberi e del campionamento delle feature da considerare quando si cerca la migliore divisione in ciascun nodo. Tale parametro viene utilizzato anche per rendere replicabile l'output del modello.

- *max_depth*: regola la profondità massima fino alla quale gli alberi all'interno della foresta possono crescere. Aumentando tale parametro, l'accuratezza del modello cresce fino ad un certo limite, ma poi inizierà a diminuire gradualmente a causa dell'overfitting nel modello.

- *min_samples_split*: specifica la quantità minima di campioni che un nodo interno deve contenere per potersi dividere in altri nodi. Se tale valore è molto basso, l'albero continuerà a crescere e comincerà a sovra-adattarsi ai dati. Aumentando il valore

del parametro, è possibile diminuire il numero totale di suddivisioni, aiutando a ridurre l'overfitting nel modello. Come per *min_samples_leaf*, il valore non dovrebbe essere troppo elevato per evitare l'underfitting.

- *criterion*: specifica il criterio per misurare la qualità di uno split. Le metriche supportate sono l'indice di Gini e l'Entropia.

Inoltre, nel presente algoritmo, sono stati aggiunti:

- *n_classes*: specifica il numero di classi in base al quale effettuare la classificazione delle osservazioni.
- *nbgenlayer*: rappresenta il numero di livelli con nodi generativi di ciascun albero; il resto dell'albero conterrà solo nodi discriminanti.
- *method_split*: contiene il metodo scelto per dividere un nodo. La divisione può avvenire casualmente o in base alla classe prevalente.
- *threshold*: viene utilizzato nei nodi generativi per scegliere la distanza dal centroide della classe in base alla quale decidere se un campione apparterrà a tale classe o meno.
- *distance*: contiene la metrica di riferimento per valutare la somiglianza tra le osservazioni delle varie classi. Le metriche supportate sono la distanza di Mahalanobis e la distanza Euclidea.

Come anticipato, una configurazione appropriata degli iperparametri è necessaria per evitare di incorrere in problemi di overfitting e underfitting (Sezione 1.3.4); essa non è mai una ricerca semplice e richiede molto spesso un'elevata complessità computazionale.

2.3.2 Vantaggi e Svantaggi

Le Random Forest presentano una serie di vantaggi e svantaggi che dovrebbero essere presi in considerazione quando le si sceglie come modello di apprendimento.
Per ciò che concerne i vantaggi si ha che:

- esse sono basate sul principio di Ensemble Learning e sulla tecnica del bagging, in modo da limitare l'overfitting negli alberi di decisione e ridurre la varianza;
- possono essere usate per risolvere sia problemi di classificazione che di regressione;
- possono gestire sia dati numerici che categorici;
- possono gestire automaticamente outlier e valori mancanti.

Gli svantaggi delle Random Forest sono, invece, i seguenti:

- *Complessità computazionale*: una Random Forest genera e combina più alberi per prendere una decisione; per questo richiede lunghi tempi di training e molta potenza di calcolo per essere implementata.
- *Interpretabilità*: la struttura più complessa di tale modello provoca maggiori difficoltà di interpretazione rispetto ai semplici alberi decisionali.

CAPITOLO 3

Il contesto di riferimento

Dopo aver mostrato una breve panoramica riguardo la teoria che è alla base del presente lavoro di tesi, in questo capitolo si vuole iniziare ad entrare nello specifico del vero e proprio lavoro sperimentale e della sua organizzazione.

3.1 Il progetto

Il progetto è stato svolto presso l'École Supérieure d'Informatique Electronique Automatique (ESIEA) [1], una università francese convenzionata con l'Università Politecnica delle Marche. Esso si colloca tra le attività svolte durante la mia esperienza Erasmus a Parigi, dal mese di settembre 2021 al mese di marzo 2022.

L'ESIEA supporta gli studenti nella realizzazione di particolari progetti, detti '*Cap Project*', proposti da aziende e laboratori di ricerca in molti campi come la Cybersecurity, l'Intelligenza Artificiale, la Data Science, la Realtà Virtuale, i Sistemi Embedded e l'Ingegneria del Software. Il progetto discusso in questa tesi è stato portato avanti da un team di 5 studenti specializzati nell'Ingegneria del Software e nell'Intelligenza Articiale. È importante precisare che esso è stato svolto come prosecuzione di un Cap Project realizzato lo scorso anno da un team di 5 studenti.

Per promuovere l'integrazione degli studenti in un ambiente rappresentativo di quello che sarà il mondo del lavoro, l'Istituto ha mobilitato un Project Management Mentor, con l'obiettivo di aiutare il team a definire le linee guida e gli obiettivi a lungo termine del progetto, a sviluppare capacità professionali e a superare le potenziali sfide che si possono incontrare in corso d'opera.

3.1.1 Il laboratorio di ricerca

Il progetto è stato svolto in partnership con il laboratorio Learning Data & Robotics (LDR), l'ente di ricerca dell'Università ESIEA. Il laboratorio raduna ricercatori in diversi campi come il Machine e il Deep Learning, i Big Data, l'Affective Computing, i Sistemi Embedded e la Robotica. Esso riunisce competenze che coprono l'intera catena di elaborazione dei dati, dall'acquisizione al processo decisionale. Il team LDR mira a creare sistemi di Intelligenza

[1] L'École Supérieure d'Informatique Electronique Automatique (ESIEA) è una Grande École d'Ingegneria specializzata in tecnologia digitale e fondata nel 1958. Oggi, la scuola ha più di mille studenti in due campus, uno a Parigi e l'altro a Laval.

Artificiale con l'obiettivo di sostenere gli esseri umani, in particolare nei campi della salute e dell'istruzione. Inoltre, il laboratorio mira a sviluppare soluzioni *frugali*, cioè che utilizzano pochi dati e potenza di calcolo, e quindi con un costo energetico inferiore.

Creato nel 2016, il laboratorio è diretto dal Prof. Lionel Prevost, Professore di ricerca presso l'ESIEA e manager del progetto discusso in questo lavoro di tesi.

3.1.2 Gli obiettivi

Una criticità degli attuali modelli di Machine Learning è che essi non si adattano facilmente ai nuovi dati per i quali non sono stati addestrati. In presenza di dati o classi sconosciute, il modello si concentra su queste nuove informazioni, "dimenticando" quanto già appreso. L'obiettivo finale di questo progetto, iniziato lo scorso anno, è far progredire lo stato dell'arte introducendo un nuovo tipo di apprendimento per le Random Forest, noto come *Incremental Learning*. L'*incrementalità* consente ad un algoritmo di apprendere e adattarsi continuamente ogniqualvolta si introducano nuovi dati, anche in una fase successiva al training iniziale. In questo modo, il modello conserva le informazioni sulle classi apprese, mentre si aggiorna per imparare quelle nuove. In particolare, il ruolo del team all'interno del progetto è stato quello di ottimizzare i parametri del modello, aggiungere nuove feature e testare le perfomance della foresta su dataset complessi.

3.1.3 Gli stakeholder

Le figure dell'Università direttamente coinvolte con il progetto sono:

- *I ricercatori del laboratorio*: la loro motivazione principale è quella di far progredire lo stato dell'arte dell'Intelligenza Artificiale.
- *Gli studenti*: da coloro che hanno inizialmente implementato il modello, ai futuri studenti che decideranno di migliorare ulteriormente l'algoritmo, essi si interessano alle nuove tecnologie e puntano ad aumentare le proprie competenze per sviluppare feature, correggere bug e manutenere il sistema.

Trattandosi di un generico algoritmo di Machine Learning che aiuta a prendere decisioni e classificare dati, esso ha numerosi campi di applicazione, come, ad esempio, la guida autonoma, la diagnosi medica e il riconoscimento emotivo. Di conseguenza, al di fuori dell'ESIEA, numerosi stakeholder condividono l'interesse per le prestazioni e i risultati dell'algoritmo. Tra questi vi sono enti di ricerca, università, imprenditori o chiunque abbia un interesse per prodotti e servizi relativi all'Intelligenza Artificiale.

3.2 Organizzazione gestionale

La realizzazione del progetto è stata divisa in due fasi. La parte iniziale è stata dedicata alla definizione di un piano di lavoro. Si tratta di un documento che contiene informazioni riguardanti il contesto di riferimento e gli obiettivi, i collegamenti con altri progetti, i ruoli e le responsabilità, i rischi e le risorse necessarie allo svolgimento del lavoro. In questa fase il team ha impiegato del tempo ad approfondire i Metodi Agili e il Framework Scrum per avere un'organizzazione ottimale e produttiva del workload. Ci si è dedicati, inoltre, all'acquisizione di conoscenze e competenze nel campo dell'Intelligenza Artificiale e del Machine Learning, e nella comprensione e interiorizzazione dell'ultima versione dell'algoritmo. Nella seconda fase si è entrati nel vivo del progetto. Con un focus principale sul miglioramento continuo, il team ha applicato a pieno il Framework Scrum e la metodologia di sviluppo software agile, rilasciando, periodicamente e in piccole porzioni, modifiche dell'algoritmo.

3.2.1 Analisi e gestione dei rischi

Il team di lavoro ha dedicato del tempo all'identificazione, all'analisi e alla gestione dei rischi, per raggiungere senza impedimenti gli obiettivi prefissati. Un approccio tipico per definire le priorità dei rischi consiste nel valutare, per ciascuno di essi, la probabilità del verificarsi dell'evento e il suo impatto. Nella Tabella 3.1, a ciascun evento dannoso viene assegnata una priorità in base alle potenziali implicazioni sugli obiettivi finali del progetto.

Descrizione	Tipo	Probabilità	Impatto	Azione preventiva	Priorità
Risorse computazionali insufficienti	Tecnico	Alta	Alto	Cloud computing	
Cattiva gestione del tempo	Organizzativo	Moderata	Alto	Pianificazione del workload secondo Scrum	
Scopo del progetto poco chiaro	Organizzativo	Moderata	Moderato	Training su AI e ML e richiesta chiarimenti al Project Manager	

Tabella 3.1: I rischi del progetto

3.2.2 Modalità di lavoro

Il carico di lavoro è stato suddiviso in diversi esperimenti da portare a termine in periodi di 1-2 settimane, detti *sprint*. Trattandosi di un progetto di ricerca e sviluppo, il workflow ha assunto la forma di un ciclo iterativo, come mostrato in Figura 3.1.

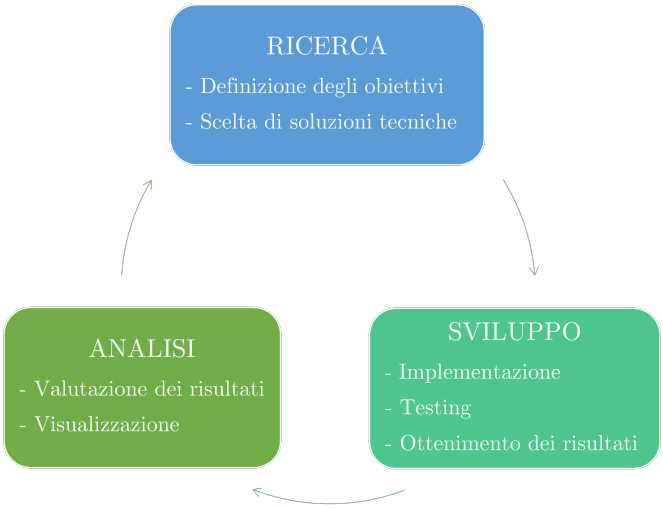


Figura 3.1: Componenti del processo iterativo

Ciascuna iterazione del ciclo era preceduta da un breve incontro con il Project Manager, allo scopo di definire l'obiettivo dello sprint, i task e la durata. Tenendo conto dei suggerimenti del Project Manager, veniva esaminata la fattibilità delle possibili soluzioni tecniche. Questa fase ha previsto sessioni di brainstorming tra i diversi membri del team, sulla base di pubblicazioni scientifiche, risorse online e ricerche bibliografiche.

La fase di sviluppo consisteva nell'implementazione della soluzione tecnica scelta in precedenza e la realizzazione di vari test per garantire il corretto funzionamento delle nuove

feature dell'algoritmo. Per migliorare la dinamicità del team, i task venivano suddivisi e distribuiti tra sottogruppi di 2 e 3 componenti di livello omogeneo. Sebbene, a causa della situazione sanitaria, questa parte del lavoro sia stata svolta in gran parte da remoto, ciò non ha influito sull'operatività e sullo spirito di collaborazione del gruppo. I membri hanno continuato a condividere giornalmente i progressi e le difficoltà incontrate, con brevi incontri su Microsoft Teams o su piattaforme di messaggistica istantanea.

L'iterazione si concludeva con la *sprint review*, un incontro in cui venivano analizzati i risultati ottenuti insieme al Project Manager, e veniva valutato l'apporto positivo o dannoso sulle prestazioni dell'algoritmo. Spesso tale incontro era seguito da un meeting con il Project Mentor, detto *sprint retrospective*, allo scopo di raccogliere feedback da ciascuno studente e di migliorare il teamwork, individuando ciò che ha o non ha funzionato.

3.2.3 Strumenti utilizzati

Il principale strumento utilizzato per il Project Management è stato GitLab; esso consiste in una piattaforma di sviluppo software open-source che offre potenti feature per la gestione e il monitoraggio di ogni stadio del ciclo di sviluppo del software. È stato scelto questo tool poiché esso concentra numerose funzionalità in un'unica piattaforma, dal controllo di versione del codice alla pianificazione degli sprint e al tracciamento dei progressi. Nella Figura 3.2 viene mostrato lo sprint planning delle prime settimane di lavoro.

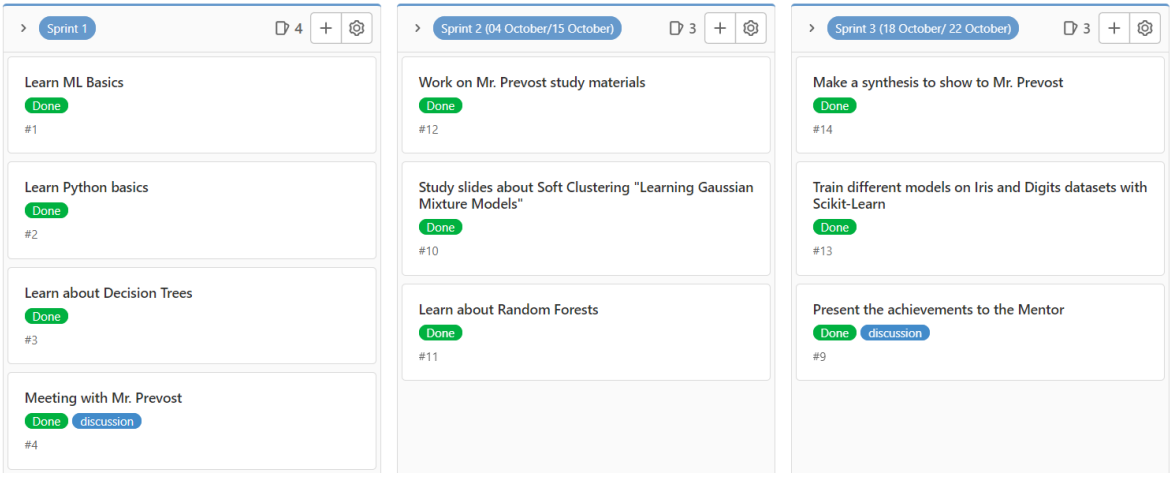


Figura 3.2: Utilizzo di GitLab per la pianificazione degli sprint

Il laboratorio di ricerca è composto da un ampio open space con uffici e sale riunioni. Per incoraggiare la condivisione di idee e informazioni tra i membri del team è stato riservato uno spazio di lavoro condiviso all'interno del laboratorio, con computer e lavagne. Tuttavia, con il peggioramento della situazione sanitaria, uno strumento fondamentale per la prosecuzione del progetto è stato Microsoft Teams; il gruppo di studenti ha utilizzato un canale della piattaforma per la comunicazione interna, e un altro canale per i meeting con il Project Manager e il Mentor.

Il codice e gli esperimenti sono stati effettuati in Python, un linguaggio di alto livello che ha il vantaggio di avere una comunità molto attiva e numerose librerie ben documentate. Attualmente, Python è il linguaggio più utilizzato nella Data Science e nell'Intelligenza Artificiale. Le librerie più utilizzate durante il progetto sono state:

- *Sklearn* per l'utilizzo di algoritmi predefiniti di Machine Learning, utili per problemi di classificazione, regressione e clustering, ma anche per la preparazione dei dati;

- *TensorFlow* per lo sviluppo di modelli di Machine Leaning che hanno bisogno di elaborare rapidamente vasti set di dati;
- *Numpy* per la manipolazione di array e matrici di dati;
- *Pandas* per la creazione e la manipolazione di dataframe;
- *Matplotlib* per la visualizzazione grafica dei dati e dei risultati.

Per semplificare la gestione e la distribuzione delle librerie, si è scelto di utilizzare Anaconda, una piattaforma di Data Science ampiamente adottata nel campo del Machine Learning in Python. L'ambiente di sviluppo integrato (IDE) scelto per modificare, effettuare il debug e testare il codice è Visual Studio Code.

Il lavoro è stato svolto combinando l'utilizzo di computer personali e macchine virtuali costruite su Google Colaboratory[2] e Google Cloud[3]. Tali macchine sono state istituite in seguito all'aumento dei requisiti di potenza di calcolo, e hanno permesso di addestrare e testare più rapidamente l'algoritmo online. La resa finale del codice è stata effettuata sul repository GitLab del laboratorio di ricerca dell'Istituto.

[2]Google Colaboratory è un ambiente Jupyter Notebook online gratuito basato su cloud, che permette di addestrare modelli di Machine e Deep Learning su CPU, GPU e TPU.

[3]Google Cloud Platform è una suite di servizi di cloud computing offerta da Google; tra i servizi offerti abbiamo l'elaborazione, l'archiviazione e l'analisi dei dati, nonché il Machine Learning.

CAPITOLO 4

Realizzazione e risultati

In questo capitolo vengono illustrati gli esperimenti svolti sulla base degli strumenti descritti nei capitoli precedenti. In primo luogo, viene esaminata l'architettura del modello e vengono descritti i dataset utilizzati per l'addestramento e per il testing. Successivamente, viene illustrata in dettaglio ciascuna sperimentazione, con tabelle e grafici riassuntivi degli impatti sulle performance dell'algoritmo.

4.1 I dataset

I dataset utilizzati nel corso del progetto sono stati principalmente 3: Iris, Digits e MNIST. Il dataset Iris fa parte della libreria Sklearn di Python e contiene 150 osservazioni di tre diversi tipi di fiori di Iris (Setosa, Versicolour, Virgnica), ognuno con 4 attributi (lunghezza e larghezza del sepalo, lunghezza e larghezza del petalo). Anche se i dati sono puliti e facili da utilizzare, questo set possiede pochi campioni per progetti di Machine Learning complessi. Esso è stato usato principalmente per testare la correttezza delle modifiche all'algoritmo, in quanto le dimensioni ridotte garantiscono tempi di calcolo minimi. Per ottenere dei risultati rilevanti sugli studi effettuati, l'algoritmo è stato testato sul dataset Digits della libreria Sklearn. Esso è utilizzato per classificare una data immagine di una cifra scritta a mano in una delle 10 classi che rappresentano valori interi da 0 a 9. Si compone di 64 feature numeriche (8×8 pixel) e una variabile target di 10 classi (0-9). Successivamente si è deciso di passare dal database Digits al database MNIST [4]. Si tratta di un dataset molto conosciuto nel campo dell'Intelligenza Artificiale, in quanto base di riferimento per confrontare le prestazioni di diversi algoritmi. Come per Digits, si tratta di riconoscere dei numeri scritti a mano (Figura 4.1).

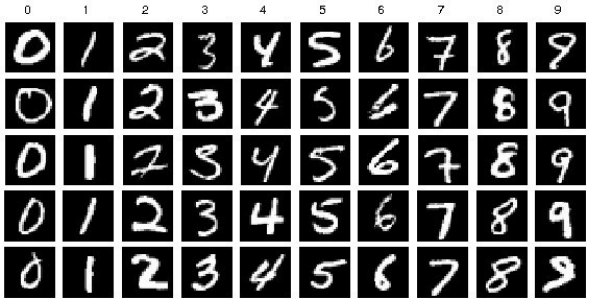


Figura 4.1: Esempi di cifre del dataset MNIST

Le immagini nel database MNIST sono in dimensioni 28x28 pixel e ciò comporta un aumento da 64 a 784 descrittori. Inoltre, mentre il database Digits contiene 1.797 campioni, MNIST ne contiene 70.000; questo comporta un aumento significativo del numero di campioni presenti e del numero di descrittori associati a ciascun campione.

I test finali dell'algoritmo sono stati effettuati su dataset di riferimento per la classificazione delle immagini, quali CIFAR-10 e Fashion-MNIST. CIFAR-10 è una raccolta di 60.000 immagini a colori in dimensioni 32x32 pixel, appartenenti a 10 diverse classi (aerei, automobili, uccelli, gatti, cervi, cani, rane, cavalli, navi e camion). Fashion-MNIST è un dataset di 70.000 immagini di articoli di Zalando, in dimensioni 28x28 pixel, e appartenenti a 10 diverse classi (t-shirt, pantaloni, pullover, abito, cappotto, sandali, camicia, sneaker, borsa e stivaletto). Le Figure 4.2 e 4.3 mostrano degli esempi di immagini contenute nei due dataset.

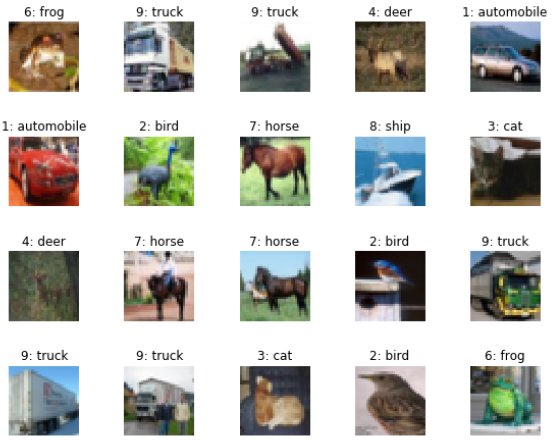


Figura 4.2: Immagini del dataset CIFAR-10

Figura 4.3: Immagini del dataset Fashion-MNIST

4.2 Il modello

Inizialmente, l'algoritmo era in grado di modellare approssimativamente la distribuzione generale di una classe; l'uso di una radice per modellare ciascuna delle classi significa assumere che la distribuzione dei dati di una classe sia unimodale. Tuttavia, le distribuzioni di probabilità corrispondenti ai dati reali sono raramente unimodali, e ciò si applica in particolare ai dati che sono stati utilizzati per testare l'algoritmo. Infatti, il dataset MNIST, su cui è stata eseguita la maggior parte dei test, è una raccolta di cifre scritte a mano. Ciascuna cifra può essere scritta in diversi modi; ad esempio, nella Figura 4.1, vengono mostrati 5 stili di scrittura di ciascun numero; essi si traducono in una distribuzione di probabilità contenente svariate modalità.

Al fine di rappresentare meglio i dati, si è passati da un modello in cui si costruisce una radice per classe, a un modello in cui si costruisce una radice per modalità. Per fare ciò, è stato necessario utilizzare un tipo di modello già noto, ovvero i *Modelli di Miscela Gaussiana* (comunemente indicati con l'acronimo inglese GMM, che sta per *Gaussian Mixture Models*). I GMM sono un algoritmo di classificazione non supervisionato (Sezione 1.2.1) che mira a trovare le distribuzioni in un set di dati. Nella implementazione dei GMM utilizzata, l'utente specifica il numero di distribuzioni da cercare; l'algoritmo cercherà di dividere il set di dati nel miglior modo possibile tra queste distribuzioni, e assegnerà a ogni campione una probabilità di appartenere a una di queste diverse rappresentazioni. Per integrare questo nuovo sistema nella Hybrid Random Forest, l'architettura dell'algoritmo ha subito delle modifiche. Nella Figura 4.4 viene mostrata la struttura dell'algoritmo prima dell'aggiunta dei GMM.

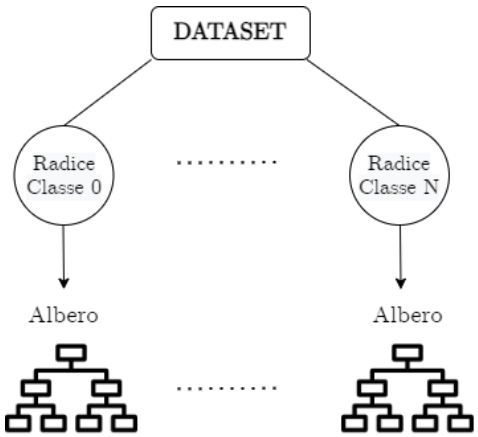


Figura 4.4: Architettura iniziale della Hybrid Random Forest

Al contrario, nell'architettura finale, invece di provare a modellare una classe per radice, il dataset viene dapprima diviso tra le varie classi. Successivamente, si ricercano le modalità di ciascuna classe utilizzando i GMM e, a seguito di ciò, si costruisce una radice per ciascuna delle modalità della classe. In entrambe le architetture, il resto dell'albero (ovvero la parte discriminante) è addestrato sull'intero dataset. Nella Figura 4.5 viene mostrata l'architettura finale dell'algoritmo.

4.3 Splitting dei nodi

Per costruire ciascun albero della foresta è necessario definire un criterio di splitting dei nodi, ovvero bisogna scegliere come decidere quali istanze andranno nel figlio sinistro e quali nel figlio destro dell'albero. L'obiettivo di questo esperimento è valutare il miglior metodo per dividere un nodo discriminante in foglie. Ci sono diversi modi per raggiungere tale

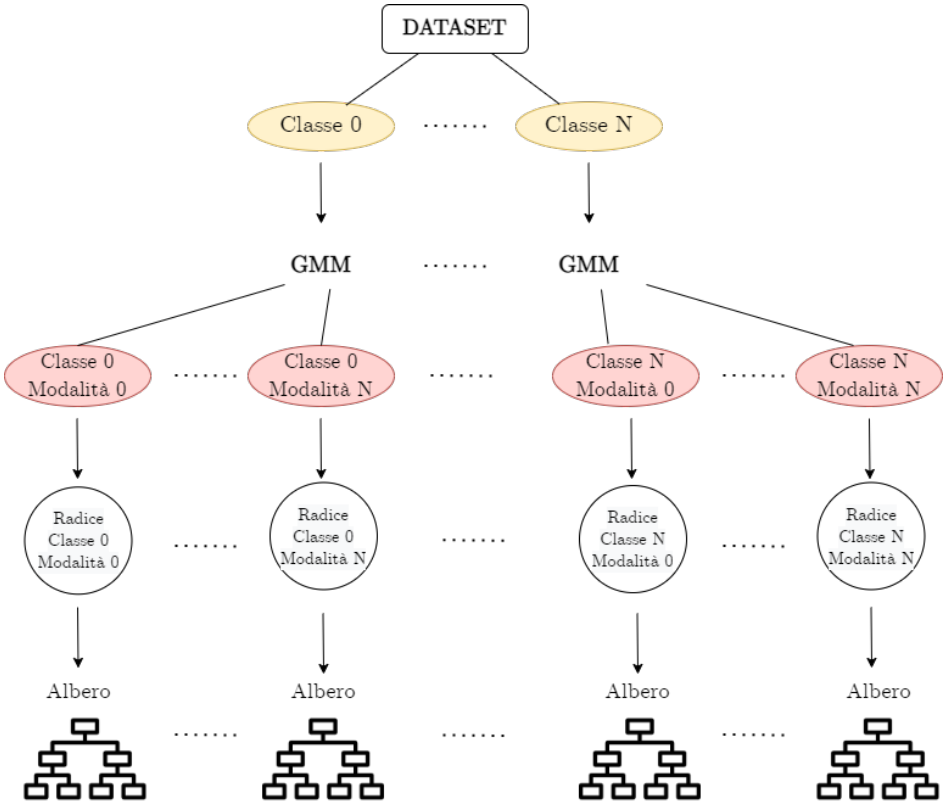


Figura 4.5: Architettura finale della Hybrid Random Forest

obiettivo; in questa sezione ne vengono discussi due: la selezione casuale e la selezione a maggioranza. La prima soluzione, detta *class bagging*, è quella utilizzata nelle Random Forest standard; essa prevede la selezione casuale di una classe all'interno del nodo corrente, e il successivo isolamento delle istanze di tale classe nel figlio sinistro dell'albero. Nella selezione della classe di maggioranza, in primo luogo si individua la classe di prevalenza nel nodo corrente. In seguito, si isolano le istanze di tale classe nel figlio sinistro dell'albero, e tutte le altre osservazioni vengono assegnate al figlio destro. Per tale esperimento sono stati definiti due valori dell'iperparametro *method_split*:

- *alea*, dove la divisione avviene isolando i campioni di una classe scelta casualmente;
- *maj_class*, dove la divisione avviene in base alla classe predominante.

Questo esperimento è stato eseguito sui dataset Wine[1], Digits e MNIST. I valori scelti per gli iperparametri della Hybrid Random Forest sono mostrati nella Tabella 4.1.

4.3.1 Risultati

Nella Tabella 4.2 sono riportati i risultati ottenuti nelle fasi di training e di testing dell'algoritmo.

Nella fase di test, le performance dell'algoritmo, utilizzando la selezione per classe di maggioranza, risultano essere lievemente migliori rispetto all'utilizzo del class bagging. Le prestazioni del modello sembrano migliorare significativamente nel caso del dataset Wine; si sospetta che ciò sia dovuto alle dimensioni ridotte del set di dati.

[1]Wine è un set di dati della libreria Sklearn di Python contenente 178 campioni di vino descritti da 13 parametri e appartenenti a 3 diverse classi.

Iperparametro	Valore
method_split	alea, maj_class
n_classes	10
n_estimators	100
min_samples_leaf	2
random_state	42
max_depth	10
min_samples_split	5
min_samples_leaf	2
max_features	sqrt
criterion	Gini
ngenlayer	0
threshold	0.95
distance	Mahalanobis

Tabella 4.1: Tuning degli iperparametri

	Training set		Test set	
	alea	*maj_class*	*alea*	*maj_class*
Wine	97.5%	96.6%	88.0%	96.6%
Digits	99.8%	99.7%	95.1%	95.5%
MNIST	97.9%	98.0%	94.7%	94.8%

Tabella 4.2: Accuratezza della foresta in base al criterio di splitting

4.4 Feature bagging

Una Random Forest può contenere centinaia di alberi, che contengono, a loro volta, centinaia di nodi. Dal punto di vista computazionale, sarebbe molto pesante esaminare tutte le feature in ogni nodo dell'albero per prendere delle decisioni; in aggiunta, in questo modo la diversità di ogni singolo albero viene meno, perdendo di vista il vero senso della foresta. È necessario, dunque, selezionare un subset casuale di feature da considerare in ogni nodo; ciò viene definito *feature bagging* (Figura 2.6). L'obiettivo di questo esperimento è individuare il numero ideale di feature da considerare in ciascuno split; ciò si traduce nella ricerca del valore ottimale per l'iperparametro *max_features* del classificatore. Tale parametro ha un impatto sul calcolo della matrice di covarianza nei nodi discriminanti, in quanto più feature vengono considerate e più parametri della matrice devono essere stimati, richiedendo maggiori risorse computazionali. Lo stato dell'arte per il numero di feature considerate in ogni nodo è *sqrt(D)*, dove D è il numero totale di feature. In una Random Forest standard, dunque, la quantità di feature considerate è fissa e uguale per ogni nodo. Tuttavia, al crescere dell'albero, il numero di campioni per nodo diminuisce; pertanto, si arriverà ad un punto in cui il numero di osservazioni nei nodi è inferiore al numero di feature considerate per descriverli. Ciò può far sì che il modello si comporti in modo inaspettato, fornisca risultati fuorvianti o abbia scarse performance su nuovi dati. Affinché l'algoritmo possa stimare correttamente i parametri di covarianza, è necessario ridurre il numero di predittori utilizzati contemporaneamente per ogni nodo al numero di campioni, cioè al crescere della profondità. In questa relazione viene

proposta una nuova formula, *divide_by_2(D)*, definita come segue:

$$\frac{D}{2^{depth}}$$

In altre parole, essa prevede di dimezzare ad ogni split il numero di feature considerate.

Nei seguenti esperimenti si confrontano i risultati ottenuti dalla formula proposta con lo stato dell'arte. I test sono stati effettuati confrontando due diverse metriche, Euclidea e di Mahalanobis, per il calcolo della distanza tra l'osservazione da classificare e la distribuzione delle classi del modello (Figura 4.6). La distanza di Mahalanobis è una metrica che tiene conto delle correlazioni tra le feature, poiché viene calcolata utilizzando l'inverso della matrice di covarianza del set di dati. In questo caso, ciascuna classe viene modellata attraverso un'ellisse. D'altra parte, l'uso della distanza Euclidea equivale ad assumere che la distribuzione dei dati sia uniforme, una circostanza che si verifica molto raramente nella realtà. Questa metrica fa uso di un cerchio per modellare i dati di ciascuna classe. Se la matrice di covarianza è la matrice identità, la distanza di Mahalanobis e quella Euclidea coincidono.

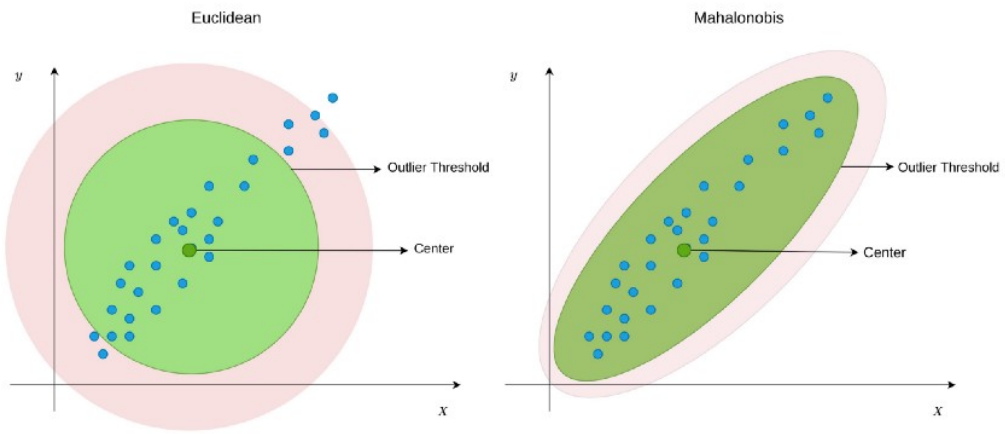


Figura 4.6: Distanza Euclidea vs distanza di Mahalanobis [2]

I test sono stati effettuati sul dataset Digits e i valori scelti per gli iperparametri della Hybrid Random Forest sono mostrati nella Tabella 4.3.

4.4.1 Risultati in funzione della profondità

La Figura 4.7 mostra il valore medio dell'impurità di Gini dei nodi ad ogni profondità della foresta, confrontando i risultati della formula proposta con lo stato dell'arte. L'indice di Gini varia tra i valori 0 e 1, dove 0 esprime la purezza della classificazione, ovvero il caso in cui tutte le osservazioni appartengono a una determinata classe, e 1 indica una distribuzione totalmente casuale degli elementi tra le varie classi. A causa dell'impatto di tale parametro sul calcolo della covarianza delle osservazioni nei nodi discriminanti, ci si aspetta che essa venga stimata con più precisione. Ciò avrà un effetto positivo sulla classificazione e, di conseguenza, sulla purezza dei nodi. Si osserva, infatti, che l'indice diminuisce più velocemente con la funzione *divide_by_2*.

Nella Figura 4.8 viene mostrato il boxplot degli indici di Gini di tutti i nodi della foresta. Si può osservare un lieve miglioramento riguardo la mediana e la varianza dell'impurità con il parametro proposto rispetto allo stato dell'arte.

Iperparametro	Valore
max_features	sqrt, divide_by_2
n_classes	10
n_estimators	100
min_samples_leaf	2
random_state	42
max_depth	10
min_samples_split	5
method_split	alea
min_samples_leaf	2
criterion	Gini
ngenlayer	0
threshold	0.95
distance	Mahalanobis, Euclidean

Tabella 4.3: Tuning degli iperparametri

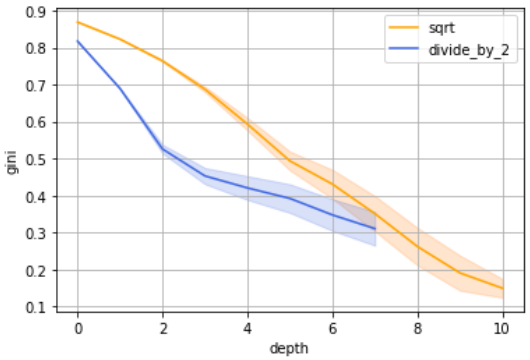


Figura 4.7: Gini per profondità

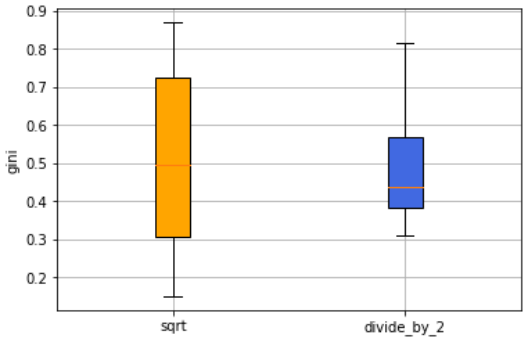


Figura 4.8: Gini per nodo

La Figura 4.9 mostra l'accuratezza media degli alberi della foresta ad ogni profondità. Si osserva che, con il parametro proposto, gli alberi raggiungono un'accuratezza migliore e più rapidamente rispetto allo stato dell'arte.

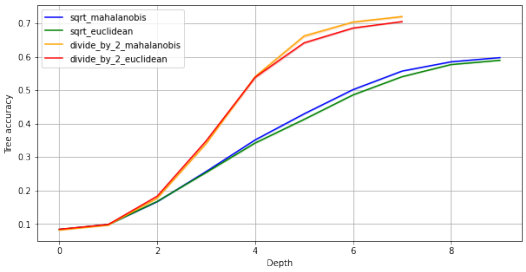


Figura 4.9: Accuratezza media degli alberi

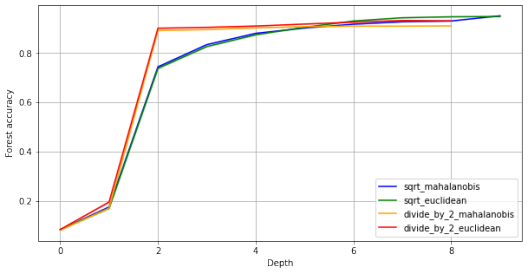


Figura 4.10: Accuratezza media della foresta

Nella Figura 4.10 è mostrata l'accuratezza media della Hybrid Random Forest per ogni profondità. Si nota che, come conseguenza di una migliore accuratezza in ciascun albero, la foresta apprende più velocemente con *divide_by_2* piuttosto che con *sqrt*. Tuttavia, ciò non comporta il raggiungimento di una maggiore accuratezza a livello complessivo.

4.4.2 Risultati globali

I risultati dei test nella Tabella 4.4 mostrano la media e la deviazione standard dell'accuratezza finale della foresta dopo 20 addestramenti.

	Training set				Test set			
	Media		Std		Media		Std	
max_features	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*
Distanza								
Mahalanobis	99.2%	100%	0.2	0.1	91.3%	93.6%	0.4	0.5
Euclidea	99.1%	99.9%	0.2	0.1	95.3%	96.4%	0.4	0.6

Tabella 4.4: Accuratezza della foresta

Si può osservare che la funzione decrescente *divide_by_2* appare, a livello di foresta, lievemente meno accurata rispetto allo stato dell'arte. D'altra parte, se si confrontano i tempi di addestramento (Tabella 4.5), si può notare che il parametro *divide_by_2* è più veloce, impiegando in media il 15% di tempo in meno rispetto alla funzione *sqrt*.

	Training set			
	Media		Std	
max_features	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*
Distanza				
Mahalanobis	34.61	41.36	1.07	2.57
Euclidea	40.97	47.04	2.70	2.95

Tabella 4.5: Tempo di addestramento (in secondi)

4.5 Threshold

Il parametro *threshold* della foresta è utilizzato nei nodi generativi per scegliere la distanza dal centroide[2] di una classe in base alla quale decidere se un campione apparterrà o meno ad essa. Come specificato nella Sezione 4.4, è possibile modellare ciascuna classe utilizzando un cerchio o un'ellisse contenente le osservazioni della classe stessa. Il threshold stabilisce la percentuale di campioni che la figura deve contenere per ottenere una modellazione ottimale di ciascuna classe; in altri termini, esso rappresenta l'estensione del raggio di queste figure.

Per un nodo generativo, il processo di riconoscimento di una classe ha diverse fasi. In primo luogo, durante la fase di training, viene calcolato il centroide di ogni classe di riferimento. Successivamente, viene calcolata la distanza tra questo centroide e ciascuno dei punti della classe ricercata. Tali distanze sono ordinate in modo crescente e, in base al threshold specificato, viene selezionata una percentuale di campioni della classe che si ritiene sia sufficiente per la modellazione. Per esempio, nel caso di una classe contenente 100 campioni, se si vuole che il 90% dei campioni rientri nel raggio, quest'ultimo viene posto pari alla 90a distanza dal centroide in ordine crescente. Applicando questo nuovo sistema, e combinandolo con la distanza di Mahalanobis, si ottengono notevoli miglioramenti nella modellazione delle classi. Sebbene inizialmente far contenere il 100% dei campioni nel raggio sembri la scelta migliore, ciò non è sempre vero. Nella Figura 4.11, viene mostrato un esempio di modellazione della classe "gatto" in base a due feature, il peso e la lunghezza del collo.

Utilizzando la distanza di Mahalanobis, la classe viene descritta attraverso un'ellisse, e il numero di campioni ivi contenuti varia in base al valore del parametro specificato. Si può

[2] Il centroide è il centro geometrico di una distribuzione di dati.

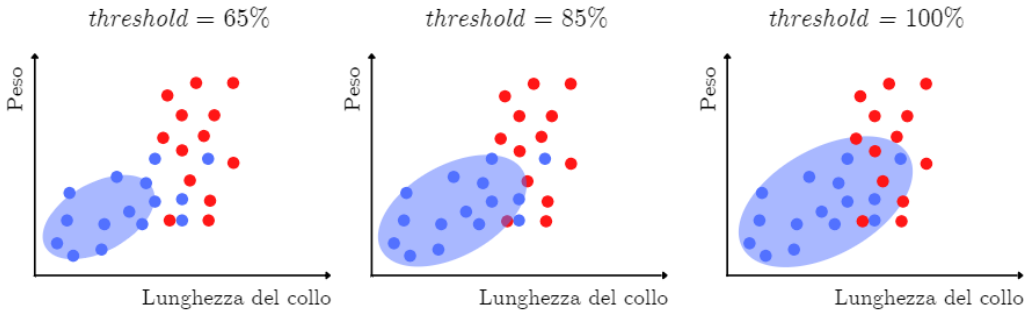


Figura 4.11: Gli effetti di diversi valori dell'iperparametro *threshold* (in blu sono rappresentate le osservazioni della classe gatto; in rosso sono rappresentate le osservazioni di altre specie animali)

notare che, se il parametro è pari a 100%, i dati della classe gatto (in blu) sono ben riconosciuti, ma i dati appartenenti ad altre specie animali (in rosso) sono inclusi nell'ellisse; pertanto, essi saranno erroneamente classificati come gatti. D'altra parte, per un valore del threshold pari al 65%, i campioni di altre specie vengono correttamente scartati, ma non tutte le istanze della classe gatto vengono riconosciute. È, quindi, necessario trovare il giusto compromesso.

Gli obiettivi di questo esperimento sono stati studiare l'influenza di tale parametro sulle prestazioni del modello e individuare un valore ottimale di tale soglia per ciascuna classe. I test sono stati effettuati sul dataset Digits e i valori scelti per gli iperparametri della Hybrid Random Forest sono mostrati nella Tabella 4.6.

Iperparametro	Valore
n_classes	10
n_estimators	100
max_depth	10
min_samples_split	5
min_samples_leaf	2
random_state	42
method_split	maj_class
min_samples_leaf	2
criterion	Gini
ngenlayer	1
max_features	divide_by_2
distance	Mahalanobis

Tabella 4.6: Tuning degli iperparametri

4.5.1 Risultati

I risultati del seguente test sono stati ottenuti facendo variare i valori del threshold per vedere se ci fosse una relazione significativa con le performance della foresta. Come mostrato nella Figura 4.12, non c'è una chiara correlazione tra l'accuratezza del modello e la soglia utilizzata nei nodi generativi. Per entrambi i dataset, un valore intorno al 70% sembra essere nel complesso la scelta ottimale, seppur i miglioramenti sono poco significativi. Sono necessari

altri esperimenti per capire se questo parametro cresce d'importanza al crescere del numero di nodi generativi.

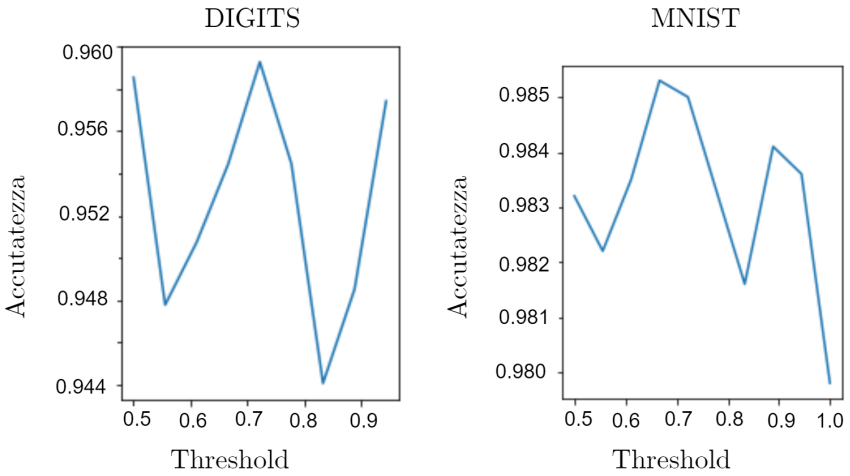


Figura 4.12: Accuratezza della foresta in funzione del *threshold*

4.6 Feature quality

In una Random Forest standard, per ciascuno split viene selezionato un sottoinsieme casuale di feature, e per ognuna di esse si calcolano gli indici di purezza per individuare i predittori che garantiscono la migliore suddivisione di un nodo. Al contrario, nella Hybrid Random Forest proposta in questa tesi, ogni nodo discriminante impiega tutte le subfeature selezionate; ciò accade poiché, nei nodi discriminanti, la classificazione si basa sul modello Nearest Class Mean (NCM). Data un'osservazione x da classificare, il classificatore NCM calcola la distanza tra il vettore delle feature di x e il valore medio dei predittori di ciascuna classe. All'osservazione x viene attribuita la classe con la media più vicina al proprio vettore delle feature, se esso è abbastanza vicino (ovvero se rientra nella soglia definita con il parametro *threshold*), altrimenti il dato viene rigettato. L'idea di questo esperimento è valutare se la selezione qualitativa delle feature possa apportare benefici all'algoritmo rispetto ad una selezione casuale. Il nuovo approccio si basa su due grandezze:

- la varianza *inter-classe*, che dipende da come la feature consente di distinguere una classe dall'altra;
- la varianza *intra-classe*, che dipende da come la feature consente di distinguere gli elementi di una classe l'uno dall'altro.

Per ciascun predittore si calcola un indice, detto *Fisher score*, definito come il rapporto tra la varianza inter-classe e la varianza intra-classe. Il risultato della selezione usando il Fisher score è una lista di feature ordinate in base alla loro importanza. Successivamente, i risultati ottenuti sono dati in ingresso a una funzione Softmax[3] che restituisce una probabilità per ogni feature: maggiore è la probabilità e più significativa risulta la feature corrispondente. In base a tale risultato è possibile effettuare una selezione qualitativa dei predittori.

I test sono stati effettuati sul dataset MNIST e i valori scelti per gli iperparametri sono mostrati nella Tabella 4.7.

[3]Softmax è una funzione che trasforma un vettore di K valori reali in un vettore di K valori reali la cui somma è pari a 1.

Iperparametro	Valore
n_classes	10
n_estimators	100
max_depth	7
min_samples_split	5
min_samples_leaf	2
random_state	42
method_split	maj_class
min_samples_leaf	2
criterion	Gini
threshold	0.95
ngenlayer	1
max_features	divide_by_2
distance	Mahalanobis

Tabella 4.7: Iperparametri utilizzati

4.6.1 Risultati

La Figura 4.13 mostra l'impurità dei nodi discriminativi di un albero a diverse profondità, confrontando la selezione casuale e la selezione qualitativa del sottoinsieme di feature. L'introduzione dell'analisi qualitativa dei predittori aiuta a raggiungere un'accuratezza paragonabile alla selezione causale ad una profondità minore dell'albero. Dalla figura si nota, infatti, che l'impurità dei nodi si riduce più rapidamente con il nuovo approccio rispetto alla selezione casuale. Si consideri, inoltre, che il computo del Fisher score per ciascuno split non ha comportato rilevanti aumenti del tempo di training.

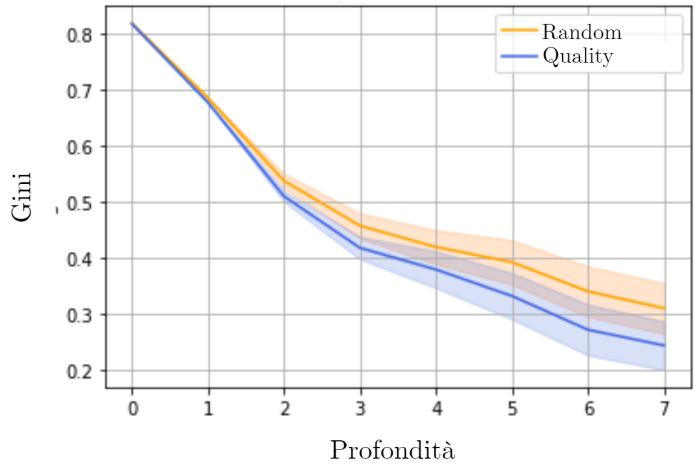


Figura 4.13: Indice di Gini di un albero della foresta a diverse profondità

4.7 Test finali

Nei test seguenti si valutano le performance dell'algoritmo su set di dati di riferimento nell'ambito della classificazione delle immagini. Trattandosi di dataset di grandi dimensioni, sono state introdotte diverse fasi di pre-elaborazione dei dati prima di alimentare il modello di apprendimento con le numerose immagini.

Fino ad ora, i descrittori utilizzati per addestrare l'algoritmo sono stati, semplicemente, i valori dei pixel delle immagini. Tuttavia, i pixel sono descrittori abbastanza scarsi, molti dei quali non contengono alcuna informazione. Infatti, nel caso di MNIST, dei 784 pixel contenuti in ciascuna immagine, 256 contengono il 98% della varianza delle istanze. La soluzione a tale problema è consistita nell'utilizzo di una Rete Neurale Convoluzionale (CNN o ConvNet), un tipo di rete neurale artificiale utilizzata per analizzare le immagini. Essa si compone di due parti: la base convoluzionale, per generare feature dalle immagini, e il classificatore, per classificare le immagini in base alle feature rilevate (Figura 4.14).

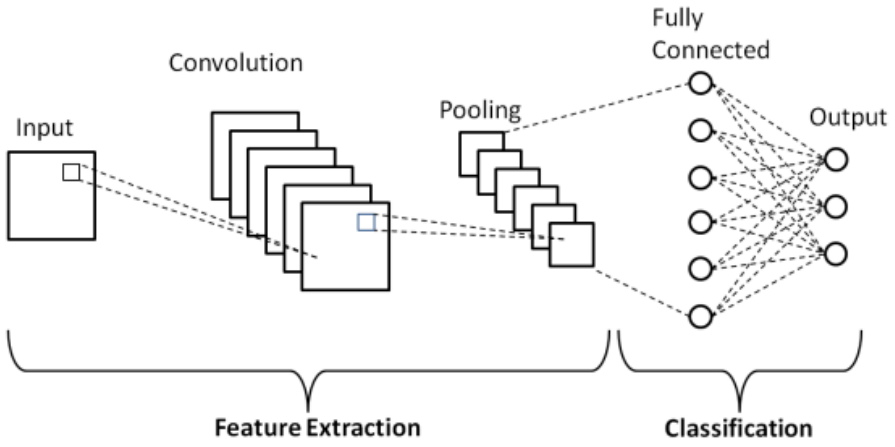


Figura 4.14: Schema dell'architettura di base di una CNN[8]

Per generare 256 descrittori e utilizzare tali feature, anziché i semplici valori dei pixel, per addestrare l'algoritmo, è stata creata una rete neurale contenente diversi strati di convoluzione, uno dei quali contiene 256 neuroni. Una volta addestrata, piuttosto che recuperare la previsione finale, questa rete viene troncata all'uscita dello strato di 256 neuroni, in modo da ottenere 256 output per ogni campione. Tali output diventeranno i descrittori utilizzati nella Hybrid Random Forest, molto più ricchi di informazioni rispetto ai pixel. In aggiunta, l'uso di una Rete Neurale Convoluzionale ha anche il vantaggio di riallineare le immagini, nel caso in cui alcune siano inclinate o capovolte.

Le Tabelle 4.8 e 4.9 mostrano i risultati ottenuti sui dataset CIFAR-10 e Fashion-MNIST. In generale, le immagini sono inizialmente passate in ingresso a una ConvNet (come DenseNet121 o AlexNet) pre-addestrata su un subset di ImageNet[3], uno tra i dataset di immagini più utilizzati. A causa dei tempi eccessivi di addestramento dell'algoritmo, non tutte le feature derivate dall'output della CNN vengono utilizzate; si è deciso, pertanto, di ridurre ulteriormente il numero di descrittori delle immagini. Questo processo di riduzione delle dimensionalità (dall'inglese *dimensionality reduction*) è basato su due diversi approcci:

- *feature selection*: si tratta di una tecnica che consiste nella scelta di un sottoinsieme di descrittori, tra quelli originali, che risultano contribuire maggiormente alla classificazione della variabile target;
- *feature extraction*: si tratta di una tecnica che consiste nell'estrazione di informazioni dal set di feature originali per creare un nuovo sottoinsieme di descrittori con una maggiore

carica semantica; a tal scopo è stata utilizzata l'Analisi delle Componenti Principali (PCA).

I descrittori che risultano da queste operazioni vengono dati in ingresso alla Hybrid Random Forest.

Modello	N° Feature	Profondità	Accuratezza (test set)	Tempo di training
DenseNet121 + PCA + NCMForest	200	1	16%	10′
		6	33%	20
		10	38%	25′
DenseNet121 + Feature Selection + NCMForest	662	1	23%	40′
		6	41%	1h
		10	46%	1h20′

Tabella 4.8: Risultati sul dataset CIFAR-10

Modello	N° Feature	Profondità	Accuratezza (test set)	Tempo di training
AlexNet + Feature Selection + NCMForest	74	1	67%	10′
		6	73%	17′
		10	75%	18′
PCA + NCMForest	74	1	70%	19′
		6	78%	20′
		10	80%	21′

Tabella 4.9: Risultati sul dataset Fashion-MNIST

Per dataset molto complessi, le performance della Hybrid Random Forest sono lievemente inferiori a quelli di una Random Forest standard; tuttavia, è possibile migliorare questi risultati potenziando l'architettura della CNN e ottimizzando ulteriormente i parametri della foresta. È importante tenere presente che la foresta raggiunge questi risultati mantenendo la possibilità di diventare incrementale, proprietà che la contraddistingue dagli altri classificatori.

CAPITOLO 5

Discussione in merito al lavoro svolto

In questo capitolo si analizza in maniera critica il progetto della presente tesi. In primo luogo, si illustrano le principali lezioni apprese durante lo svolgimento del progetto. Successivamente, si discutono i punti di forza del lavoro svolto e le parti che potrebbero essere migliorate in futuro.

5.1 Premessa

Il lavoro svolto nella presente tesi ha permesso di acquisire esperienza riguardo le metodologie adottate per il Project Management e per lo sviluppo agile del software e ha consentito di sperimentare diversi tool nell'ambito del Machine Learning. A conclusione di un progetto è necessario fare un resoconto critico di quanto realizzato, in modo che il team possa attingere a questo bagaglio di conoscenze. Ciò è importante per non ripetere gli errori fatti in passato e riproporre le strategie che hanno funzionato.

Per quanto riguarda le principali lezioni apprese durante il lavoro svolto, esse sono state riorganizzate e raccolte nei seguenti punti:

- monitoraggio degli esperimenti;
- definizione di un obiettivo di ricerca chiaro e di traguardi intermedi.

Ad ogni lezione dedicheremo una sottosezione del seguente paragrafo.

5.2 Lezioni apprese

5.2.1 Monitoraggio degli esperimenti

Nell'apprendimento automatico, il monitoraggio degli esperimenti è un fattore chiave per ottenere risultati consistenti. Tenere traccia del preprocessing dei dati, dei valori degli iperparametri o degli ambienti di sviluppo utilizzati è fondamentale per riuscire a stabilire con esattezza come i risultati siano stati prodotti. In particolare, i metadati degli esperimenti sono importanti per:

- allineare e condividere i risultati con il resto del team;
- riprodurre gli esperimenti;
- interpretare gli esiti dei test.

In primo luogo, è necessario accordarsi tra i membri del team sui valori di alcuni iperparametri (ad esempio *random_state*), cosicché i risultati ottenuti possano essere confrontati. Inoltre, registrare i valori dei parametri per ogni esperimento è importante per poter interpretare, migliorare e riprodurre quanto ottenuto. A tale scopo, una buona pratica consiste nel tracciare il processo di elaborazione dei dati utilizzati. Nel mondo reale, i dati contengono generalmente outlier e valori mancanti e presentano formati inadatti ad essere utilizzati direttamente per i modelli di apprendimento automatico. Il preprocessing è un'attività necessaria per pulire e rendere i dati adatti a un modello di Machine Learning. In questa fase, ad esempio, i dati etichettati in modo errato vengono rimossi, si migliorano le feature e vengono stabilite le proporzioni tra Training set, Validation set e Test set.

5.2.2 Definizione di un obiettivo di ricerca chiaro e di traguardi intermedi

La definizione degli obiettivi di ricerca è fondamentale per condurre un progetto con successo. Molti team si trovano in difficoltà poiché all'inizio non si dedica abbastanza tempo alla definizione adeguata dello scopo del lavoro e, di conseguenza, si sprecano risorse preziose per raccogliere informazioni irrilevanti o non necessarie. Le *best practice* da adottare per definire gli obiettivi di ricerca sono:

1. Ascoltare le richieste del Project Manager e discutere gli obiettivi generali da raggiungere con la ricerca.
2. Documentarsi sul contesto di riferimento e chiedere chiarimenti relativi a dubbi o ambiguità sul progetto.
3. Suddividere lo scopo centrale della ricerca in fasi più piccole e task specifici.
4. Utilizzare la tecnica SMART per creare obiettivi definiti, verificabili e raggiungibili. SMART è un acronimo inglese che indica le 5 qualità fondamentali che un obiettivo deve possedere, ovvero:
    - *Specific*: i task devono essere scritti in modo chiaro e vanno risolte nell'immediato eventuali ambiguità.
    - *Measurable*: è importante stabilire le metriche per misurare i progressi che conducono al raggiungimento degli obiettivi.
    - *Achievable*: gli obiettivi definiti devono essere raggiungibili per evitare di essere sopraffatti da aspettative irrealistiche.
    - *Relevant*: gli obiettivi devono essere rilevanti per la ricerca al fine di restare motivati e di proseguire sulla buona strada.
    - *Time-based*: per la buona riuscita del lavoro è fondamentale fissare una scadenza principale per l'intero progetto e termini più ravvicinati per ciascun obiettivo.

5.3 Punti di forza e di debolezza dell'approccio realizzato

In questa sezione si analizzano i punti di forza e i punti di debolezza del lavoro svolto. Questa valutazione è utile per comprendere quali sono realmente i "cavalli di battaglia" del progetto e, d'altra parte, quali sono gli aspetti meno curati che potrebbero essere migliorati.

5.3.1 Punti di forza

Il punto di forza principale dell'algoritmo è il modello utilizzato, dal momento che esso si basa su un approccio innovativo alla classificazione dei dati che si avvicina all'apprendimento umano, con il fine ultimo di riprodurre la capacità di imparare nel corso della vita e di stratificare la conoscenza.

Un altro caposaldo del sistema riguarda il contributo dell'algoritmo alla comunità scientifica. Il laboratorio LDR prevede di divulgare i risultati ottenuti attraverso una pubblicazione scientifica, in modo da rendere l'algoritmo una fonte di ispirazione per altri progetti nell'ambito del Machine Learning.

Un punto di forza dell'algoritmo è che, sebbene sia stato addestrato su un dataset di cifre scritte a mano, esso può essere facilmente adattato per la classificazione di tipi di dati differenti e per diverse applicazioni. Infine, un aspetto importante del lavoro svolto è la comprensibilità del codice. Grazie ad una continua attività di refactoring, il codice finale è pulito, comprensibile e ben commentato, in modo da facilitare la manutenibilità del sistema e la possibilità di dare in carico il progetto ad altri studenti in futuro.

5.3.2 Punti di debolezza

Guardando con un occhio critico il sistema realizzato, è possibile senz'altro individuare degli aspetti migliorabili.

Un primo punto di debolezza riguarda la complessità computazionale dell'algoritmo. L'intero processo risulta essere piuttosto dispendioso sia in termini di risorse che di tempo, in particolare per la fase di training, che può richiedere ore. Per quanto riguarda il testing, grazie all'utilizzo del Transfer Learning mediante Reti Neurali e delle tecniche di riduzione delle dimensionalità, si è riusciti a ridurre notevolmente i tempi di esecuzione, senza dover scendere a compromessi sulle performance dell'algoritmo.

Un aspetto da migliorare è la capacità di riconoscere le classi apprese e individuare potenziali nuovi raggruppamenti grazie alle radici generative. L'algoritmo mette a disposizione diversi approcci per l'apprendimento incrementale. Quando viene individuata una nuova classe, è possibile scegliere tra l'aggiornamento delle foglie dell'albero, l'introduzione di nuovi nodi o la ridefinizione delle funzioni di splitting dei nodi. Quest'ultimo approccio produce risultati più che soddisfacenti; tuttavia, tale operazione comporta la necessità di addestrare nuovamente intere parti degli alberi, che può avere un costo computazionale molto elevato per alberi di grandi dimensioni. L'approccio ideale sarebbe quello di riaddestrare esclusivamente i nodi in cui avviene lo splitting, in modo da ridurre notevolmente il tempo di training.

Un altro aspetto implementabile in maniera differente riguarda il calcolo della matrice di covarianza. A volte, seppur raramente, si verifica un errore del tipo *"SVD did not converge"*, in riferimento all'algoritmo *Singular Value Decomposition* utilizzato per l'inversione delle matrici di covarianza. Per evitare ciò, si è pensato di utilizzare un valore di tolleranza da aggiungere ai valori della matrice prima dell'inversione, per far sì che essi siano sempre non nulli e non generino eccezioni. Tuttavia, si ritiene che il calcolo delle matrici con gli strumenti di Sklearn, come la covarianza ridotta, la covarianza inversa sparsa o la covarianza robusta, possa risolvere in modo definitivo questi problemi.

CAPITOLO 6

Conclusioni

In questo lavoro di tesi è stata presentata una nuova tipologia di apprendimento per le Random Forest, detto *incremental learning*, che consente all'algoritmo di imparare e adattarsi continuamente, ogni volta che emergono nuovi dati, anche dopo il training iniziale. Dai risultati sperimentali ottenuti, si può concludere che l'algoritmo compete con lo stato dell'arte attuale nell'ambito delle Random Forest. Non tutti i risultati ottenuti sono eccellenti, ma sono, comunque, incoraggianti: è importante tenere presente che la foresta raggiunge tali performance mantenendo la possibilità di diventare incrementale, una proprietà che la contraddistingue dagli altri classificatori. Tali risultati devono essere considerati come un punto di partenza per ampliare e migliorare l'algoritmo. Per esempio, in futuro, sarebbe possibile ampliare la quantità e la tipologia di dati disponibili per addestrare il modello, affinare le tecniche di estrazione delle feature o studiare la possibilità di introdurre un approccio più "performante" per l'apprendimento incrementale.

Con questo lavoro si è voluto mettere in risalto come l'Intelligenza Artificiale sia ormai utilizzata in modo pervasivo in un'ampia gamma di applicazioni e con approcci sempre più vicini al modo umano di apprendere.

Bibliografia

[1] BENTO, C. (2021), «Decision Trees nella vita reale», URL https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575. (Citato a pag. iv e 13)

[2] CANSIZ, S. (2021), «Multivariate Outlier Detection in Python», URL https://towardsdatascience.com/multivariate-outlier-detection-in-python-e946cfc843b3. (Citato a pag. iv e 29)

[3] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K. e FEI-FEI, L. (2009), «Imagenet: A large-scale hierarchical image database», in «2009 IEEE conference on computer vision and pattern recognition», p. 248–255, Ieee. (Citato a pag. 35)

[4] DENG, L. (2012), «The mnist database of handwritten digit images for machine learning research», *IEEE Signal Processing Magazine*, vol. 29 (6), p. 141–142. (Citato a pag. 24)

[5] MAYRAZ, G. e HINTON, G. (2001), «Recognizing Hand-written Digits Using Hierarchical Products of Experts», vol. 13.

[6] MININI, A. (2022), «Gli alberi di decisione», URL https://www.andreaminini.com/ai/machine-learning/alberi-di-decisione. (Citato a pag. 12)

[7] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M. e DUCHESNAY, E. (2011), «Scikit-learn: Machine Learning in Python», *Journal of Machine Learning Research*, vol. 12, p. 2825–2830. (Citato a pag. 17)

[8] PHUNG e RHEE (2019), «A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets», *Applied Sciences*, vol. 9, p. 4500. (Citato a pag. iv e 35)

[9] RASCHKA, S. (2015), *Python Machine Learning*, cap. 1,2, Packt Publishing. (Citato a pag. 6)

[10] SIMPLILEARN (2022), «Classification in Machine Learning: What it is and Classification Models», URL https://www.simplilearn.com/tutorials/machine-learning-tutorial/classification-in-machine-learning#what_is_classification. (Citato a pag. 10)

[11] WIKIPEDIA, L'ENCICLOPEDIA LIBERA (2022), «Apprendimento dell'albero decisionale», URL `https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=1082660125`. (Citato a pag. 13)

Siti consultati

- `www.towardsdatascience.com`

Ringraziamenti

Desidero ringraziare il mio docente e relatore Prof. Domenico Ursino, per aver accolto con entusiasmo la mia scelta di svolgere la tesi in collaborazione con un ateneo francese, e soprattutto per aver dimostrato ogni giorno la sua straordinaria disponibilità. Oltre ad avermi guidato nella stesura di questo lavoro, mi ha trasmesso fin da subito la passione e l'entusiasmo per questa professione.

Ringrazio i partner francesi, per avermi dato l'opportunità di realizzare questo progetto, supportandomi nella parte implementativa del lavoro con preziosi consigli e spiegazioni fondamentali.

Un ringraziamento speciale va alla mia famiglia: è grazie al sostegno, ai sacrifici e all'incoraggiamento dei miei familiari se sono riuscita a raggiungere questo traguardo. A loro dedico questa tesi, perché sono loro la mia più grande fonte di ispirazione.

Grazie ai miei amici, ai miei compagni di corso e alla mia coinquilina Asia per aver alleggerito le lezioni, le attese agli appelli e le ansie pre-esame. Grazie per i mercoledì sera e le serate memorabili; senza di voi questa triennale sarebbe stata senz'altro più pesante!

Grazie in particolare a Gioele e Riccardo, per la gentilezza con cui mi hanno trattata nei progetti svolti insieme, per le risate davanti ai caffè e per le chiacchierate fino a tardi.

Un ringraziamento speciale va a Niccolò, per avermi ascoltata e sostenuta nei momenti di difficoltà e per aver gioito con me dei miei traguardi. Sono certa che la nostra è una di quelle amicizie che supererà il tempo e le distanze.

Infine, ringrazio di cuore Maristella e Sara, le mie compagne di avventure: ciò che c'è tra noi è la conferma che l'amicizia, quella vera, esiste. Grazie per i viaggi, per le confidenze, per avermi fatto forza e per essermi state accanto sempre, da ogni parte del mondo. Vi voglio bene.