



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Comparazione algoritmi di Machine Learning per la classificazione di immagini

RELATORE

Prof. Fabio Palomba

CORRELATORE

Dottorando **Giammaria Giordano**

Università degli Studi di Salerno

CANDIDATO

Stefano Biddau

Matricola: 0512105824

“Se vuoi qualcosa vai e inseguila.” (Will Smith)

Sommario

L'Intelligenza Artificiale si sta diffondendo in molti campi offrendo vari contesti applicativi. Il riconoscimento e la classificazione delle immagini è ciò che ne consente molti dei risultati più impressionanti. Lo scopo di questa ricerca è quello di considerare i più famosi algoritmi di intelligenza artificiale nell' ambito della classificazione quali KNN, Alberi Decisionali, Random Forest, SVM e anche le Reti Neurali Artificiali e di testare la loro efficienza nel classificare le immagini campione raccolte nei dataset MNIST e F-MNIST. La ricerca è stata condotta attraverso l'utilizzo di librerie Python come Scikit-learn, che mettono a disposizione questi algoritmi e permettono di effettuare un lavoro di settaggio dei loro parametri, individuando quelli che più incidono sulla loro capacità di apprendere, al fine di migliorare la loro accuratezza nella classificazione quando questi ultimi vengono addestrati su dataset forniti. In base al loro comportamento si è potuto fornire una valida motivazione al perché del passaggio di testimone che è avvenuto dal dataset MNIST, ormai ritenuto obsoleto e abusato, a F-MNIST suo successore. Si è potuto notare infatti che gli algoritmi sopra citati a parità di parametri considerati, mostrano un tasso di accuratezza nella classificazione minore su F-MNIST rispetto a MNIST. Ciò evidenzia una maggiore difficoltà dell'uno (F-MNIST) rispetto all'altro (MNIST) e di conseguenza un'esigenza di considerare più a fondo caratteristiche di tali algoritmi in modo da renderli sempre più accurati nel loro compito. Di fatto le ottime performance che tali algoritmi offrono addestrandosi su F-MNIST ne garantiscono risultati efficienti indubbiamente su tutti gli altri dataset esistenti.

Link alla repository GitHub del lavoro svolto: <https://github.com/stefanBid/Classificazione-Immagini>

Indice	ii
Elenco delle figure	v
Elenco delle tabelle	vii
1 Introduzione	1
1.1 Motivazioni e Obiettivi	2
1.2 Risultati	2
1.3 Struttura della tesi	2
2 Background e Stato dell'Arte	4
2.1 Background	4
2.1.1 Machine Learning	4
2.1.2 Classificazione delle immagini	5
2.1.3 Dataset	6
2.1.4 Algoritmi di intelligenza artificiale	12
2.1.5 Overfitting e Underfitting	12
2.1.6 K-Nearest Neighbors (KNN)	14
2.1.7 Alberi Decisionali	17
2.1.8 Random Forest	19
2.1.9 Macchina a vettori di supporto (SVM)	22
2.1.10 Rete Neurale Artificiale (ANN)	25

2.2	Stato dell'Arte	29
2.2.1	Riconoscimento delle cifre scritte a mano	29
2.2.2	Sistema di riconoscimento ottico dei caratteri di cifre scritte a mano . .	31
2.2.3	Classificazione F-MNIST basata sul descrittore delle caratteristiche HOG	33
2.2.4	Addestramento efficiente di alberi decisionali robusti con GROOT . .	35
2.2.5	Limiti sullo Stato dell'Arte	38
3	Metodologia di ricerca	39
3.1	Obiettivo della ricerca	39
3.2	Materiali	39
3.3	Metodi	40
3.3.1	Operazioni preliminari alla classificazione	40
3.3.2	Scelta degli iper-parametri	42
3.3.3	Simulazione e addestramento effettivo	45
3.3.4	Punti chiave dell'approccio metodologico all' esperimento	47
4	Risultati	49
4.1	Risultati classificazione con k-Nearest Neighbors	49
4.1.1	KNN su MNIST	49
4.1.2	KNN su F-MNIST	51
4.2	Risultati classificazione con Alberi Decisionali	53
4.2.1	Alberi Decisionali su MNIST	53
4.2.2	Alberi Decisionali su F-MNIST	55
4.3	Risultati classificazione con Random Forest	57
4.3.1	Random Forest su MNIST	57
4.3.2	Random Forest su F-MNIST	59
4.4	Risultati simulazione di classificazione con SVM	61
4.4.1	SVM su MNIST	62
4.4.2	SVM su F-MNIST	64
4.5	Risultati classificazione con Percettrone Multinastro	66
4.5.1	Percettrone Multinastro su MNIST	66
4.5.2	Percettrone Multinastro su F-MNIST	68
4.6	Confronto finale	70

5 Conclusioni	72
5.1 Lavoro svolto	72
5.2 Risultati ottenuti	73
5.3 Sviluppi Futuri	74
Ringraziamenti	75

Elenco delle figure

2.1	Esempio Database MNIST	7
2.2	Grafico indici di popolarità dei dataset	8
2.3	Struttura Fashion-MNIST	10
2.4	Diagramma del processo di conversione utilizzato per generare il dataset Fashion-MNIST.	11
2.5	Modelli di classificazione binaria messi a confronto (<i>underfitting</i> , <i>overfitting</i> , bilanciato)	13
2.6	Processo di classificazione di un campione tramite algoritmo KNN	15
2.7	Processo di classificazione di un campione tramite algoritmo DT	18
2.8	Processo di classificazione di un campione tramite algoritmo RF	21
2.9	Processo di classificazione attraverso la Macchina a Vettori di Supporto	23
2.10	Struttura del neurone	25
2.11	Struttura di una VNN	26
2.12	Struttura di una DNN	26
2.13	Campione di 8 cifre scritte a mano utilizzato per testare i classificatori KNN, SVM e MPL	30
2.14	Risultati di classificazione tramite SVM e MPL messi a confronto	31
2.15	Schema di estrazione delle caratteristiche in base alla zona	32
2.16	Matrice di confusione dei risultati ottenuti dalla classificazione delle cifre scritte a mano con MPL	33
2.17	Caratteristiche estratte da un immagine campione di F-MNIST	34
2.18	Matrice di confusione delle caratteristiche HOG su F-MNIST	35

2.19 Risultati di classificazione messi a confronto	37
3.1 Grafico a torta ripartizione dataset MNIST/F-MNIST nei due set di train e test	41
3.2 Schema riconoscimento underfitting/overfitting	47
3.3 Grafico catena metodologica script 1 per analisi delle prestazioni di un classificatore generico	48
3.4 Grafico catena metodologica script 2 per analisi delle prestazioni di un classificatore generico	48
4.1 Risultati simulazione addestramento KNN su MNIST	50
4.2 Risultati grafici addestramento effettivo KNN su MNIST	51
4.3 Risultati simulazione addestramento KNN su F-MNIST	52
4.4 Risultati grafici addestramento effettivo KNN su F-MNIST	53
4.5 Risultati simulazione addestramento Alberi decisionali su MNIST	54
4.6 Risultati grafici addestramento effettivo Alberi Decisionali su MNIST	55
4.7 Risultati simulazione addestramento Alberi decisionali su F-MNIST	56
4.8 Risultati grafici addestramento effettivo Alberi Decisionali su F-MNIST	57
4.9 Risultati simulazione addestramento Random Forest su MNIST	58
4.10 Risultati grafici addestramento effettivo Random Forest su MNIST	59
4.11 Risultati simulazione addestramento Random Forest su F-MNIST	60
4.12 Risultati grafici addestramento effettivo Random Forest su F-MNIST	61
4.13 Output grafici Simulazione addestramento SVM su MNIST	63
4.14 Output grafici Simulazione addestramento SVM su F-MNIST	65
4.15 Parte dell'output testuale del primo script	66
4.16 Output grafico del primo script (profondità rete)	66
4.17 Risultati grafici addestramento effettivo Random Forest su MNIST	68
4.18 Risultati grafici simulazione addestramento percettrone multinastro su F-MNIST	69
4.19 Risultati grafici addestramento effettivo percettone multinastro su F-MNIST	70
5.1 Percentuali di accuratezza dei vari algoritmi messe a confronto	73

Elenco delle tabelle

2.1	Assegnamento etichette alle classi di F-MNIST	9
2.2	Risultati esperimento di classificazione di cifre scritte a mano con KNN	30
2.3	Risultati di accuratezza sul set di test di MNIST e F-MNIST messi a confronto	38
4.1	Tabella dei migliori tre risultati di addestramento KNN su MNIST	50
4.2	Tabella dei migliori tre risultati di addestramento KNN su F-MNIST	52
4.3	Tabella dei migliori tre risultati di addestramento Alberi Decisionali su MNIST	54
4.4	Tabella dei migliori tre risultati di addestramento Alberi Decisionali su F-MNIST	56
4.5	Tabella dei migliori tre risultati di addestramento Random Forest su MNIST .	58
4.6	Tabella dei migliori tre risultati di addestramento Random Forest su F-MNIST	60
4.7	Tabella dei migliori risultati per funzione del kernel SVM su MNIST	62
4.8	Tabella dei migliori risultati per funzione del kernel SVM su F-MNIST	64
4.9	Tabella dei migliori tre risultati di addestramento percettrone multinastro su MNIST	67
4.10	Tabella dei migliori tre risultati di addestramento percettrone multinastro su F-MNIST	69
4.11	Tabella riassuntiva dei risultati	71

CAPITOLO 1

Introduzione

Al giorno d'oggi l'uso dell'intelligenza artificiale è sempre più frequente in molteplici ambiti. La possibilità che le macchine acquisiscano un qualsiasi tipo di ragionamento o imparino dagli errori offre la possibilità di ottenere risultati più rapidamente e quindi svolgere compiti che alla portata umana richiederebbero moltissimo tempo. Sempre più aziende si servono di essa per studiare metodi e tecniche che permettono di progettare sistemi sia hardware che software che migliorino il risultato del lavoro.

Il processo di apprendimento è molto simile a quello umano, gli algoritmi devono essere prima istruiti per poi poter lavorare in completa autonomia e garantire la sicurezza che essi apprendano realmente e non inizino ad un certo punto a imparare sistematicamente.

L'intelligenza artificiale è un potente strumento, ad esempio dà la possibilità di applicare la capacità di apprendimento ad un sistema informatico costruito attraverso un linguaggio di programmazione e quindi di per sé orientato a svolgere operazioni sistematicamente. Una volta fatto ciò, questo sistema può essere utilizzato per svolgere molteplici compiti, uno di questi è sicuramente la capacità di saper riconoscere delle immagini, e saperle classificare.

Per fare ciò vi sono a disposizione delle grandi banche dati che forniscono agli algoritmi delle proprietà che li aiutano in una fase di training a distinguere un tipo di campione da un altro. La particolarità è che questi dataset sono universali, quindi permettono di addestrare tutti i tipi di algoritmi da quello più specifico a quello più generico. Tra quelli più utilizzati oggi vi sono MNIST e il suo successore F-MNIST, dei dataset caratterizzati da una serie di campioni di immagini.

1.1 Motivazioni e Obiettivi

Questo lavoro di tesi si prefigge 3 obiettivi principali:

1. Capire come l'intelligenza artificiale permette agli algoritmi di essere in grado di riconoscere delle immagini e soprattutto di saperle distinguere effettuando un'operazione di classificazione;
2. Individuare gli insiemi di iper-parametri più importanti che incidono sull'accuratezza di classificazione di questi algoritmi;
3. Fornire una risposta a quale tra i due dataset proposti (MNIST e F-MINIST) aiuti in maniera più efficace gli algoritmi in questo compito.

1.2 Risultati

I risultati ottenuti da questa ricerca hanno evidenziato l'importanza di combinare più iper-parametri (tra principali e secondari) al fine di rendere il compito di classificazione di un algoritmo di intelligenza artificiale più accurato e specifico, fornendo così ampi margini di miglioramento nel campo della classificazione di ogni genere. Inoltre parallelamente a questo studio la ricerca si è concentrata anche su un altro punto fondamentale: l'esigenza di individuare un dataset di addestramento robusto e che costituisca un banco di prova efficiente per preparare gli algoritmi alla classificazione. Dallo studio è emerso che il tra i due dataset analizzati quello più efficiente risulta essere F-MNIST sul quale gli algoritmi mostrano avere una percentuale di accuratezza leggermente più bassa, ma tutto ciò è dovuto a una maggiore complessità dei campioni (in termini di pixel) che caratterizzano questo set di dati e che quindi lo rendono un banco di prova più affidabile per i test.

1.3 Struttura della tesi

Le tematiche sopra menzionate sono state ampiamente discusse nei successivi capitoli. Precisamente la tesi è strutturata nel seguente modo:

- **Capitolo 1 (Introduzione):** tratta le motivazioni che hanno portato allo sviluppo di questa ricerca ed i vari obiettivi che si prefigge;
- **Capitolo 2 (Background e Stato dell'Arte):** tratta le nozioni di base e le esperienze fatte in letteratura utili a comprendere diversi aspetti della ricerca effettuata;

- **Capitolo 3 (Metodologia di ricerca):** illustra la metodologia applicativa della ricerca effettuata;
- **Capitolo 4 (Risultati):** descrive quali risultati sono stati ottenuti dalla ricerca effettuata;
- **Capitolo 5 (Conclusioni):** spiega in che modo la ricerca possa essere utilizzata ed i suoi possibili sviluppi futuri.

CAPITOLO 2

Background e Stato dell'Arte

Questo capitolo è composto da due sezioni, nella prima vengono mostrati i concetti fondamentali associati all'apprendimento automatico. Questi saranno utili a comprendere i lavori svolti in letteratura, mostrati nella seconda sezione del capitolo, oltre che per comprendere l'esperimento mostrato nel capitolo successivo.

2.1 Background

2.1.1 Machine Learning

Che cos'è il Machine Learning ?

Quando parliamo di Machine Learning ci riferiamo ad un ramo specifico dell'informatica associato all'intelligenza artificiale. In un recente articolo, A.Longo [1] asserisce che non è sempre possibile definire le proprietà e le applicazioni del machine learning in modo semplice, anche per via del suo campo molto vasto che offre una molteplicità di metodi, tecniche e strumenti di implementazione. In aggiunta, le diverse tecniche di apprendimento e algoritmi di sviluppo causano tante possibilità di utilizzo quante sono le finalità, rendendo difficile una definizione precisa. Ad ogni modo, parlando di Machine Learning, parliamo di macchine intelligenti che migliorano le loro abilità e prestazioni nel tempo tramite diversi meccanismi. Le macchine riescono ad imparare a compiere compiti specifici migliorando le loro abilità e reazioni attraverso l'esperienza. Alla base del ML c'è una serie di diversi algoritmi che, da conoscenze primitive, prenderanno decisioni o andranno a compiere azioni apprese nel tempo.

I diversi apprendimenti di una macchina

Ancora, nell'articolo di A. Longo [1] possiamo vedere come egli si concentra sullo sviluppo del settore e di come è stato costante in questi anni ad avere apportato modalità di apprendimento diverse, efficaci, non solo per gli algoritmi utilizzati, ma anche per la finalità per cui le macchine stesse sono state realizzate. In base a come la macchina apprende ed accumula dati e informazioni, si possono osservare tre sistemi di apprendimento automatico:

- **Apprendimento supervisionato;**
- Apprendimento non-supervisionato;
- Apprendimento per rinforzo.

I lavori in letteratura che seguono e l'esperimento si basano sull'apprendimento supervisionato.

Apprendimento supervisionato

Questa tecnica consiste nel fornire il sistema informatico della macchina di una serie di modelli ed esempi che permettono di costruire un vero e proprio database (teoricamente **dataset**) di informazioni ed esperienze, in modo che quando la macchina avrà davanti a sé un problema, dovrà solo basarsi sulle esperienze registrate nei dati, analizzarle e decidere quale risposta dare alle esperienze già codificate. Si può affermare che questa tipologia di apprendimento è in qualche modo già condizionata, cosicché la macchina necessita solo di essere in grado di rispondere nel modo più adeguato allo stimolo dato. Gli algoritmi che utilizzano l'apprendimento supervisionato sono utili in particolar modo per poter risolvere problemi di classificazione.

2.1.2 Classificazione delle immagini

Nel 2021 Ihal [2] sottolinea come il riconoscimento e la classificazione delle immagini consentono molti dei risultati più impressionanti dell'intelligenza artificiale, riferendosi ad esempio a come un telefono può determinare cos'è un oggetto semplicemente scattandone una foto, oppure in che modo i siti web dei social media taggano automaticamente le persone nelle foto. I metodi di classificazione delle immagini possono essere ampiamente suddivisi in due categorie: classificazione basata su pixel e classificazione basata sugli oggetti.

Un pixel è l'unità di base di un'immagine e l'analisi dei pixel è il metodo principale per eseguire la classificazione dell'immagine. Tuttavia, un algoritmo di classificazione può

classificare un’immagine utilizzando solo le informazioni spettrali nei singoli pixel esaminando le informazioni spaziali insieme alle informazioni spettrali. I metodi di classificazione basati sui pixel utilizzano solo informazioni spettrali (intensità dei pixel), mentre i metodi di classificazione basati su oggetti considerano sia le informazioni spettrali che spaziali dei pixel.

2.1.3 Dataset

Nel paragrafo precedente abbiamo visto che il dataset è uno dei componenti fondamentali per consentire l’apprendimento supervisionato. Un dataset è un insieme o collezione di dati omogenei, informazioni numeriche, testo, immagini o suoni che ci permettono di risolvere problemi di intelligenza artificiale e questi dati possono essere suddivisi in classi in base a determinati attributi comuni. La scelta del dataset è fondamentale in relazione al campo del problema da trattare, nel caso della classificazione delle immagini quelli utilizzati sono: MNIST e Fashion-MNIST

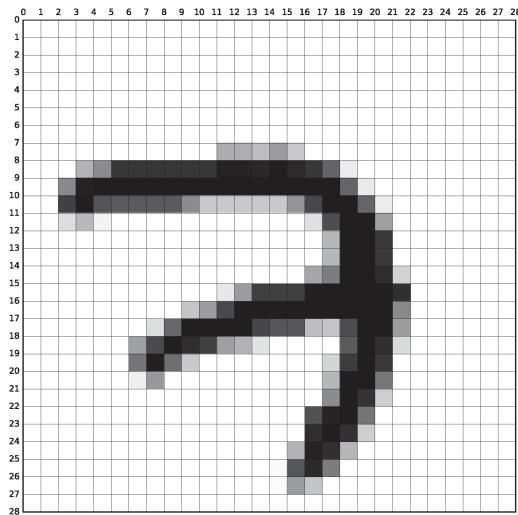
MNIST

Il dataset MNIST, composto da 10 classi di cifre scritte a mano, è stato introdotto per la prima volta da LeCun et al [3] nel 1998. Il database MNIST, di loro realizzazione, contiene un totale di 70.000 istanze di cui 60.000 per l’addestramento e il resto per il test. Questo database è composto da due diverse fonti: Special Database 1 del NIST¹ e Special Database 3 del NIST². I set di addestramento e i test sono stati scelti in modo che lo stesso autore non fosse coinvolto in entrambi i set. Il set di formazione include esempi di oltre 250 autori e le immagini originali sono state sottoposte a una pre-elaborazione. Questa procedura ha comportato innanzitutto la normalizzazione delle immagini per farle rientrare in un riquadro di 20×20 pixel, preservando il rapporto d’aspetto, poi è stato applicato un filtro anti-aliasing e le immagini in bianco e nero sono state trasformate in scala di grigi. Infine, è stato introdotto un padding vuoto per inserire le immagini in un riquadro superiore a 28×28 pixel, in modo che il centro di massa della cifra corrispondesse al suo centro. 2.1a viene mostrato un esempio di campione corrispondente alla cifra "7" dopo che è stato sottoposto alla pre-elaborazione

¹Contiene 2.100 immagini a pagina intera di campioni di scrittura a mano stampati da 2.100 diversi studenti geograficamente distribuiti negli Stati Uniti

²Contiene i dati del campione, ovvero le informazioni raccolte dalle domande poste a un campione di tutte le persone e unità abitative, recuperato del Census Bureau

sopra citata. Mentre, nella figura 2.1b si mostra un insieme più ampio di istanze appartenenti al set di addestramento, suddivise per classi di appartenenza (ogni classe occupa una riga).



(a) MNIST campione appartenente alla cifra 7



(b) 100 campioni dal set di addestramento MNIST

Figura 2.1: Esempio Database MNIST

MNIST è uno dei dataset più importanti realizzati fino ad oggi, ed è un riferimento per vari problemi di classificazione. A sostegno di quanto detto Xiao [4] afferma che i membri delle comunità A.I.³, M.L. e Data Science⁴ esaltano questo set di dati e lo usano come benchmark per convalidare i propri algoritmi. In effetti, MNIST è spesso il primo set di dati provato dai ricercatori perché costituisce un buon banco di prova per testare l'apprendimento degli algoritmi sancendone da subito una prima efficacia, che può essere eventualmente contraddetta se l'algoritmo viene testato su qualche altro dataset più complesso.

Fashion-MNIST

Il dataset Fashion-MNIST nasce come sostituto del tradizionale MNIST per diverse ragioni, tra le principali:

- **MNIST è troppo semplice** infatti le reti convoluzionali possono raggiungere il 99,70% su MNIST. Gli algoritmi di apprendimento automatico classici possono anche raggiungere facilmente il 97,00%. Per ulteriori informazioni si rimanda al capitolo 2;
- **MNIST è abusato** infatti Ian Goodfellow, ricercatore di Google Brain ed esperto di deep learning in un suo thread su Twitter mostra come dagli anni 2000 ad oggi l'utilizzo di

³Intelligenza Artificiale: ampia area dell'informatica il cui compito è associare l'intelligenza umana alle macchine.

⁴Scienza focalizzata sulla scoperta di informazioni ottenibili da grandi insiemi di dati grezzi strutturati e non.

MNIST sia cresciuto esponenzialmente superando di gran lunga qualsiasi altro dataset realizzato come mostrato in Figura 2.2.

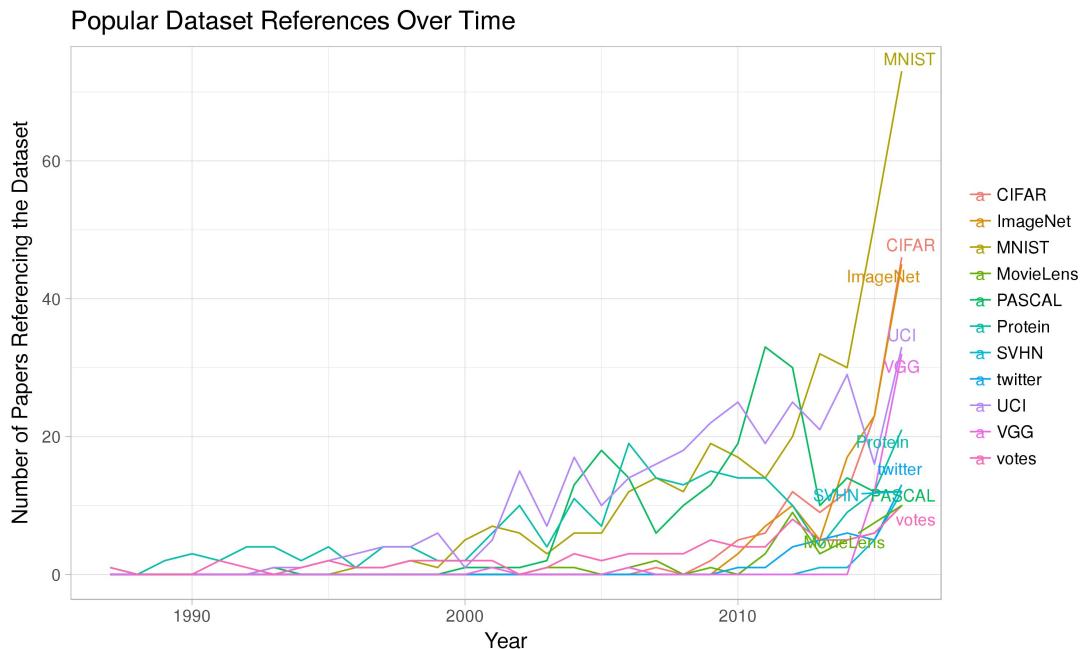


Figura 2.2: Grafico indici di popolarità dei dataset

Il set di dati Fashion-MNIST è stato presentato da Zalando Research⁵ (Xiao et al., 2017 [4]) e si basa sull’assortimento presente sul sito web di Zalando⁶. Ogni prodotto di moda su Zalando ha una serie di immagini scattate da fotografi professionisti, che mostrano diversi aspetti del prodotto, ad esempio il fronte e il retro, i dettagli o come appare indossato da una modella/o. L’immagine originale ha uno sfondo grigio chiaro (colore esadecimale: #fdfdfd) e viene salvata in formato JPEG 762x1000, dopodiché viene campionata nuovamente con diverse risoluzioni, ad esempio grande, media, piccola, miniatura e minuscola.

⁵Lanciato nel 2016 è il dipartimento duraturo ed esplorativo dell’azienda che riunisce esperti interni, scienziati e sviluppatori <https://tech.zalando.com>

⁶La più grande piattaforma di moda online d’Europa <https://www.zalando.com>

Fashion-MNIST è stato creato con 70.000 miniature di prodotti unici, questi prodotti appartengono a diversi gruppi di genere come uomini, donne, bambini e persone neutre. I prodotti bianchi in particolare non sono inclusi nel set di dati perché hanno un basso contrasto con lo sfondo. Ad ogni istanza di addestramento e test è assegnata un'etichetta. L'intera associazione è mostrata nella Tabella 2.1.

Etichetta	Descrizione
0	T-shirt/top
1	Pantalone
2	Maglione
3	Vestito
4	Cappotto
5	Sandalo
6	Camicia
7	Sneaker
8	Borsa
9	Stivaletto

Tabella 2.1: Assegnamento etichette alle classi di F-MNIST

In Figura 2.3 si mostra un esempio di come appaiono i dati in Fashion-MNIST. Ogni classe di abbigliamento occupa 3 righe della matrice, si parte dalla classe di abbigliamento "0" che corrisponde a "T-shirt/top", per finire con la classe di abbigliamento "9" che corrisponde a "Stivaletto" (per un totale di 30 righe).

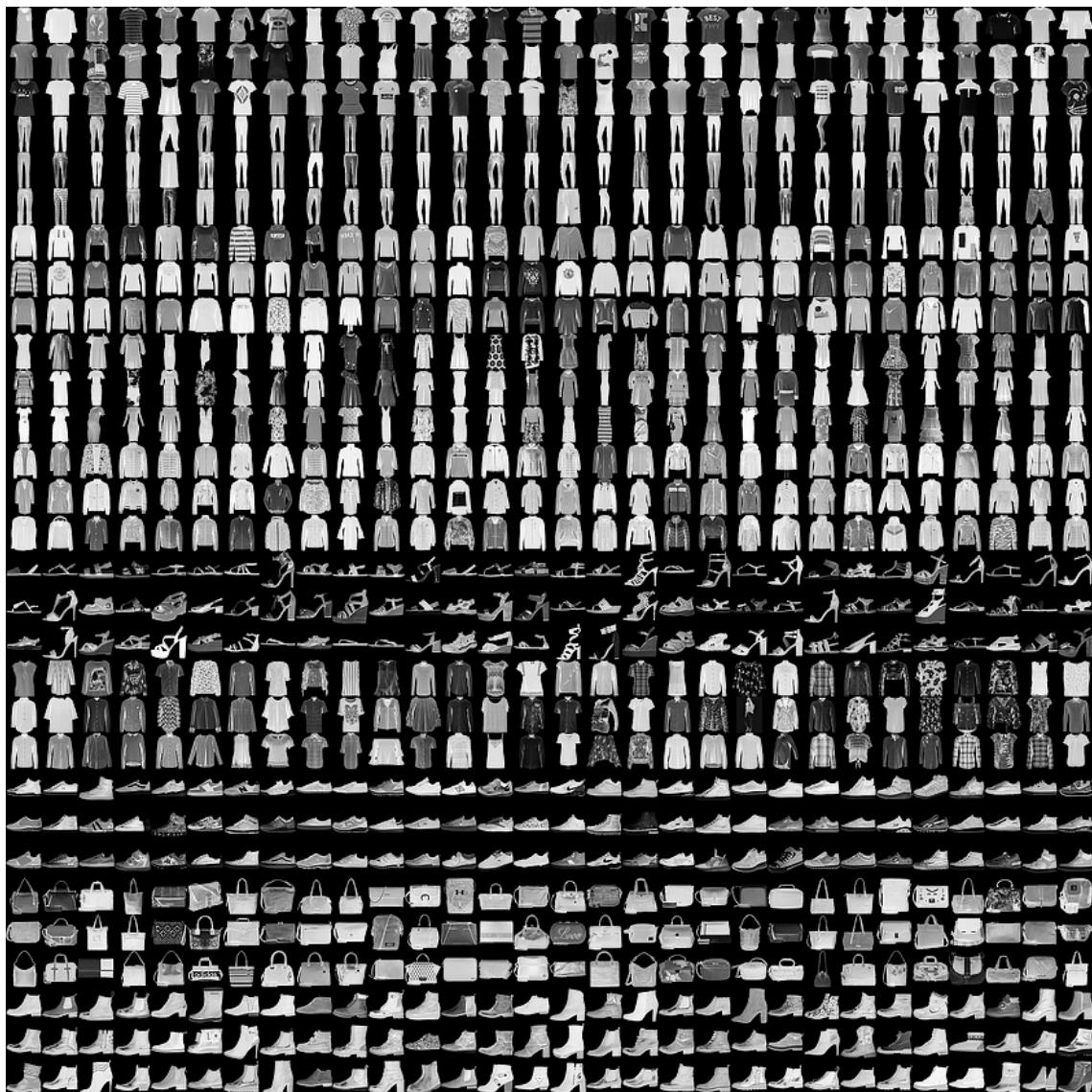


Figura 2.3: Struttura Fashion-MNIST

Come per MNIST, anche le immagini di F-MNIST sono sottoposte ad un processo di pre-elaborazione prima di essere inserite nel dataset. Di fatto le istanze si ottengono a partire dalle miniature (51x73) che vengono inviate alla seguente catena di conversione:

1. Conversione dell'input in un'immagine PNG;
2. Ritaglio bordi vicini al colore dei pixel d'angolo (La "vicinanza" è definita dalla distanza entro il 5% dell'intensità massima possibile nello spazio RGB);
3. Ridimensionamento del bordo più lungo dell'immagine a 28 mediante sotto campionamento dei pixel (ciò avviene saltando alcune righe e colonne);
4. Aumento nitidezza dei pixel grazie ad un operatore gaussiano di raggio e deviazione standard pari a 1,0, con effetto crescente in prossimità dei contorni;
5. Estensione del bordo più corto a 28 e posizionamento dell'immagine al centro della tela;
6. Negazione intensità dell'immagine;
7. Conversione dell'immagine in pixel in scala di grigi a 8 bit.

Tale processo di conversione viene mostrato chiaramente in Figura 2.4 dove abbiamo due campioni di esempio:

- Un'istanza della classe di abbigliamento "3" che corrisponde ad un "Vestito";
- Un'istanza della classe di abbigliamento "5" che corrisponde ad un "Sandalo".

Entrambe le immagini subiscono il processo di conversione, e la figura mostra passo per passo l'output visivo di ogni fase.

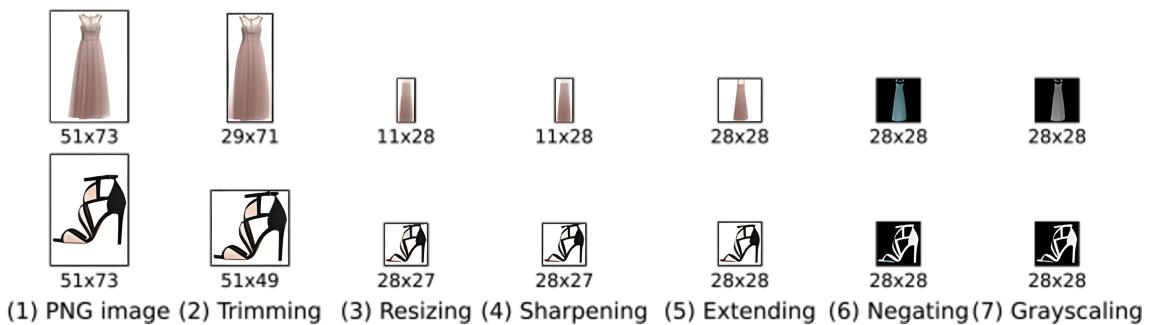


Figura 2.4: Diagramma del processo di conversione utilizzato per generare il dataset Fashion-MNIST.

2.1.4 Algoritmi di intelligenza artificiale

Un ulteriore elemento fondamentale dell'apprendimento automatico è l'insieme degli algoritmi di intelligenza artificiale che permettono di addestrare la macchina la quale riuscirà, successivamente, ad effettuare predizioni. Possiamo trovare molti algoritmi di intelligenza artificiale, diversi tra di loro per molti fattori: meccanismo di funzionamento, performance, tempo di addestramento del modello, sono solo alcuni dei criteri con il quale tali algoritmi si suddividono, ma sono anche i principali con i quali si sceglie uno piuttosto che un altro per effettuare una determinata classificazione.

2.1.5 Overfitting e Underfitting

Valutare l'efficienza e l'accuratezza di un classificatore è un lavoro che deve tener conto di due fattori fondamentali:

1. **Overfitting:** è un problema che si presenta quando la classificazione si basa su troppi parametri. In questi casi, la varianza della classificazione diventa elevata, perché il modello risulta eccessivamente sensibile ai dati addestramento;
2. **Underfitting:** è un problema di apprendimento che si verifica quando la classificazione si basa su pochi parametri. La classificazione soffre di un'eccessiva discrepanza di conseguenza risulta troppo semplice.

In linea di massima, si desidera selezionare un modello che sia una via di mezzo tra underfitting e overfitting. In Figura 2.5 si compara la classificazione di due classi diverse (rossa e verde) da parte di un modello che soffre di underfitting, uno che soffre di overfitting e uno bilanciato. Come si può vedere il primo modello effettua una classificazione in cui è molto facile esprimere la variazione. Nel secondo invece la variazione è molto marcata e precisa sintomo di un modello che ha imparato sistematicamente, il terzo invece quello bilanciato mostra una linea di separazione corretta e con la giusta varianza.

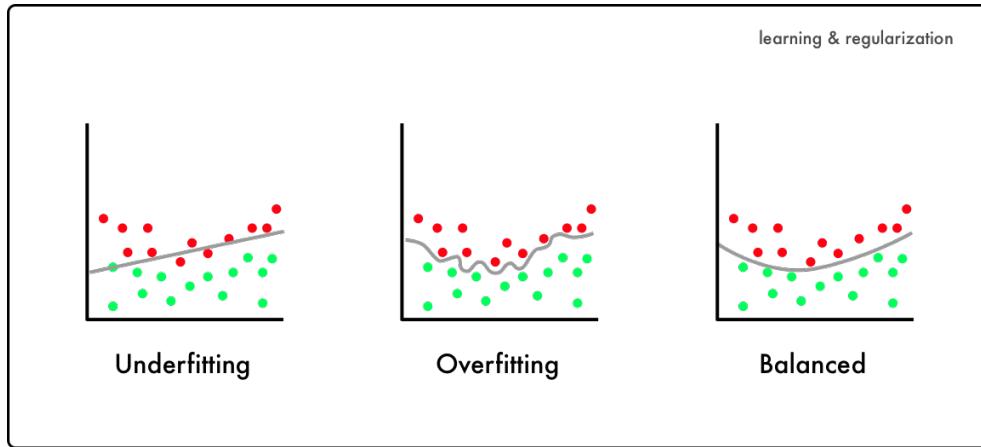


Figura 2.5: Modelli di classificazione binaria messi a confronto (underfitting, overfitting, bilanciato)

Esistono due tecniche importanti che si possono utilizzare durante la valutazione degli algoritmi di apprendimento automatico per limitare l'overfitting e l'underfitting:

- Utilizzare una tecnica di ricampionamento per stimare l'accuratezza del modello;
- Trattenere un set di dati di convalida.

In questo lavoro di tesi ci si è concentrati sulla prima. La tecnica di ricampionamento più popolare è la convalida incrociata k-fold: essa consente di addestrare e testare il modello k-volte su diversi sottoinsiemi di parametri di addestramento e di creare una stima delle prestazioni del modello di apprendimento automatico.

2.1.6 K-Nearest Neighbors (KNN)

L'algoritmo k-nearest neighbors noto anche come KNN , è un classificatore di apprendimento supervisionato non parametrico che usa la prossimità per effettuare classificazioni o previsioni sul raggruppamento di un singolo punto di dati. Nonostante possa essere utilizzato per problemi di classificazione o regressione viene principalmente utilizzato per le problematiche di prima tipologia, basandosi sul presupposto che punti simili possono trovarsi vicino.

Funzionamento

In un articolo sul proprio sito, IBM⁷ [5] approfondisce il principio di funzionamento di tale algoritmo per classificazione e regressione. Per quanto riguarda i problemi di classificazione un'etichetta di classe viene assegnata sulla base di un voto a maggioranza, ovvero viene utilizzata l'etichetta più frequente rappresentata attorno un preciso numero di dati. Se ci sono solo due categorie la scelta ricadrà sull'etichetta che per maggioranza supererà il 50%; la soglia che permetterà la scelta si abbassa quando le categorie aumentano.

In Figura 2.6 si mostra chiaramente quanto detto prima. Viene mostrata la rappresentazione grafica dei campioni appartenenti a due classi(classe A e classe B), e seguendo la numerazione di ogni grafico possiamo vedere i vari step che portano alla classificazione di un nuovo campione:

1. Il punto rosso corrisponde al nuovo campione da classificare;
2. L'algoritmo tiene conto prima di classificare tale elemento dei suoi vicini e dalla distanza che hanno da esso;
3. Si individuano quelli che realmente entrano in gioco nel processo decisionale (2 classe A, 3 classe B);
4. Per un voto di maggioranza il nuovo campione viene etichettato alla classe B.

⁷International Business Machines Corporation, è l'azienda statunitense più anziana e tra le maggiori al mondo nel settore informatico <https://www.ibm.com/it-it>

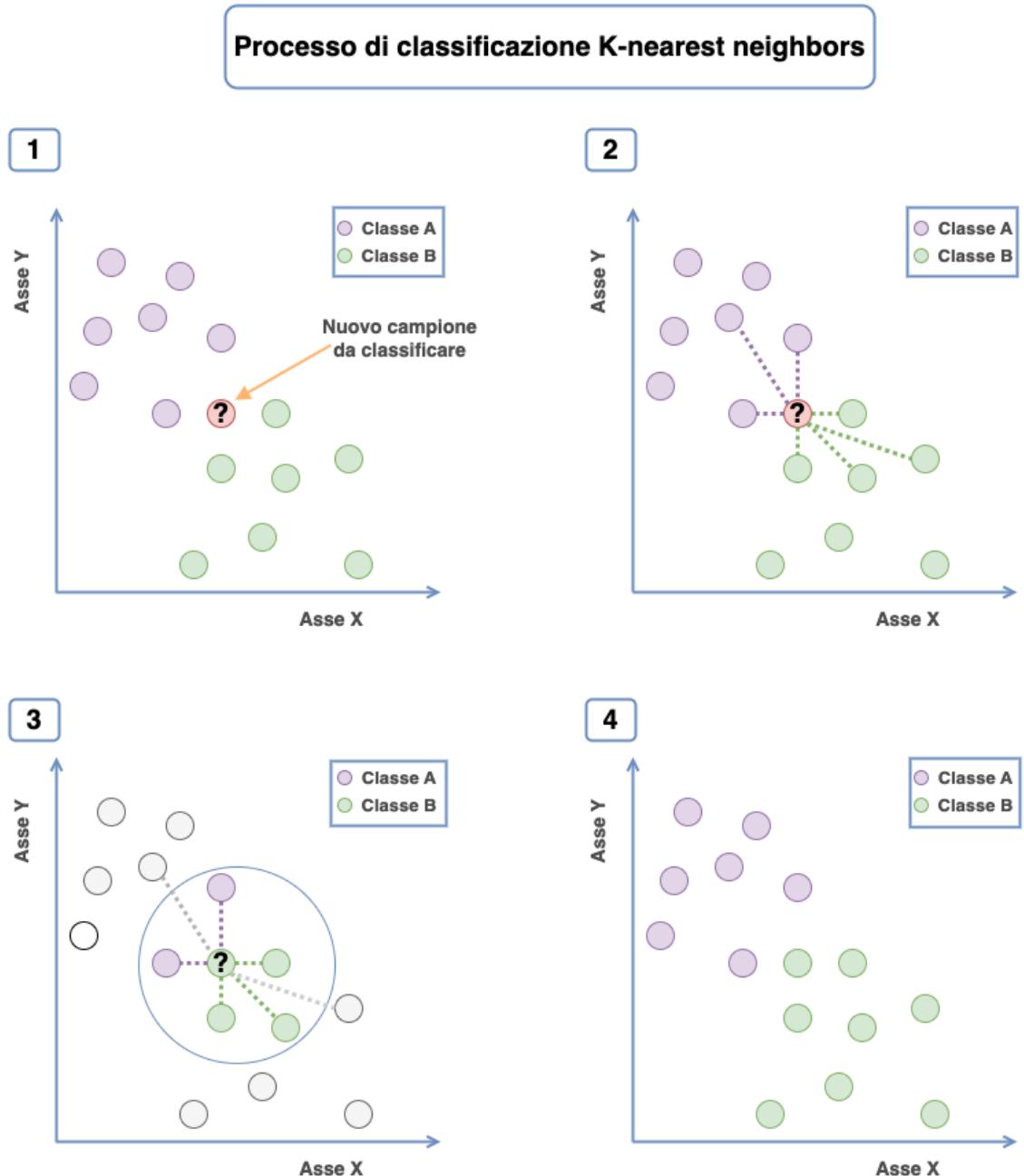


Figura 2.6: Processo di classificazione di un campione tramite algoritmo KNN

Parametri del modello

Il primo parametro da considerare è la **metrica di distanza** che aiuta a formare i limiti decisionali tra un punto di query e gli altri punti dati. Esistono diverse tipologie, le più importanti sono:

- **euclidea:** misura la linea retta tra il punto di query e un altro punto che si sta misurando;
- **manhattan:** misura il valore assoluto tra due punti;

- **minkowski**: questa misura della distanza è la forma generalizzata delle metriche di distanza euclidea e di manhattan.

Altro parametro molto importante che incide sulle prestazioni del classificatore è il **valore k** che è associato al numero di vicini che verranno controllati per determinare la classificazione di un punto specifico di query; ad esempio se $k = 1$, l'istanza verrà assegnata alla stessa classe del suo singolo vicino più vicino e definire k può essere un atto di bilanciamento in quanto valori diversi possono portare ad overfitting o underfitting; la determinazione di questo parametro dipende solo ed esclusivamente dai dati in input del dataset. Per convenzione si utilizzano principalmente dei valori dispari per evitare pareggi nella classificazione.

Infine ci sono i pesi (**weights**), si riferiscono alla funzione peso utilizzata per prevedere i possibili valori. Possono essere di due tipi:

1. **uniforme**: tutti i punti in ogni vicinato hanno lo stesso peso e di conseguenza la stessa influenza;
2. **distanza**: in questo caso i punti più vicini al punto da classificare influenzano più dei punti più lontani.

Pro e contro

PRO:

- Facile da implementare data la sua semplicità e allo stesso tempo forte accuratezza;
- Si adatta facilmente quando vengono aggiunti nuovi campioni;
- Pochi iper parametri rispetto ad altri algoritmi di Machine Learning.

CONTRO:

- Non ha una buona scalabilità essendo un algoritmo pigro, occupa più memoria e spazio di Storage rispetto agli altri classificatori;
- Non funziona molto bene con input di dati ad alta dimensione.

2.1.7 Alberi Decisionali

L'albero decisionale è un algoritmo di tipo supervisionato usato nel ML. Questo è un modello predittivo che può essere utilizzato sia per i casi di classificazione che di regressione: ciò significa che l'output che andrà a prevedere il modello sarà una variabile categorica oppure una quantità continua. Gli alberi decisionali sono gli algoritmi più comuni e implementati perché sono molto veloci e, soprattutto, semplici da utilizzare e interpretare. Questo è fondamentale perché in alcuni settori una grande semplicità di comprensione può essere preferita ad una grande precisione del modello.

Funzionamento

Come scrive D.Nardini [6], in un suo articolo nel 2021, in questo algoritmo i dati che si danno come input vengono continuamente splittati in base a determinati criteri. Per capire il loro funzionamento possiamo individuare nei nodi e nelle foglie due concetti chiave.

- **I nodi:** sono i luoghi in cui in base a certe regole i dati vengono splittati;
- **Le foglie:** sono i risultati intermedi o finali ossia i luoghi in cui finiscono i dati una volta splittati.

In Figura 2.7 si comprende nello specifico quanto detto prima. Nel grafico a sinistra viene mostrata la rappresentazione grafica dei campioni appartenenti a due classi (classe A e classe B), mentre a destra un possibile processo decisionale di un albero. Come si può osservare, lo split degli elementi avviene in base a dei valori di due proprietà; se i campioni rispettano o meno i valori di tali proprietà vengono splittati. Alla fine con una profondità dell'albero pari a 4 si riesce a catalogare i campioni in modo ottimale. Difatti, come possiamo vedere sul grafico si creano 5 sezioni al cui interno abbiamo campioni appartenenti tutti alla stessa classe.

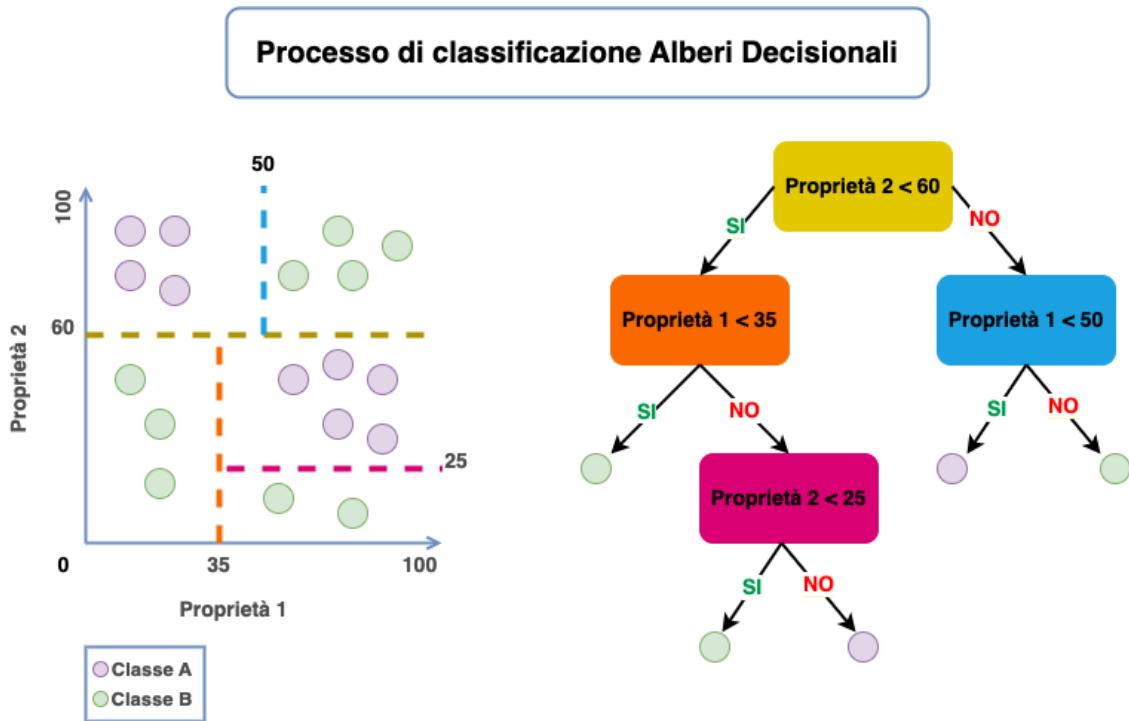


Figura 2.7: Processo di classificazione di un campione tramite algoritmo DT

Parametri del modello

Per prima cosa c'è la **metrica dello split**, che separa i dati misurando la qualità dello split. Questo parametro è chiamato **criterion** e dipende dall'attività che si desidera eseguire. Per quanto riguarda il problema della classificazione, i suoi possibili valori possono essere:

- **gini**: fa riferimento all'impurità di Gini;
- **entropy**: fa riferimento all'informazione gain, basato sul concetto di entropia nella teoria dell'informazione.

Altro parametro molto importante è **max-depth**, questo indica quanto è profondo l'albero, più è profondo, più si divide e più preciso è nei dati di addestramento. Tuttavia, ciò non vuol dire che se è più preciso sui dati di train il modello sarà in grado di generalizzarli bene, anzi più sarà profondo, più preciso sarà sui dati di train, e meno probabilmente sarà in grado di generalizzare su dati nuovi, ciononostante una maggior profondità quindi rischia di far cadere il modello in overfitting.

Un altro parametro che può portare overfitting è **min-samples-leaf**, ovvero quanti esemplari minimi di osservazioni possono finire nelle foglie finali. Un numero basso vorrà dire anche in questo caso una maggior precisione sul train, che potrà portare di nuovo ad una

scarsa accuratezza sul test. Ciò accade perché si andranno ad identificare i casi più specifici e non quelli generali.

Infine vi è **max-features**: questo parametro ci dà la possibilità di scegliere un numero di variabili da utilizzare per trovare il miglior split. Di default lo split verrà cercato su tutte le variabili. Tuttavia, si può anche scegliere di trovare il miglior split sulla radice quadrata del numero delle variabili (se ne ho 16, lo cerco su 4).

Pro e contro

PRO:

- Molto semplice da capire e interpretare;
- Può lavorare su dati numerici e categorici contemporaneamente;
- Non richiede moltissimi dati;
- Performa bene sia sui pochi dati che su grandi dataset.

CONTRO:

- Non è un modello robusto, quindi un piccolo cambiamento nel training può comportare un gradissimo cambiamento nella predizione;
- Altissimo rischio di over-fitting.

2.1.8 Random Forest

Uno degli algoritmi più utilizzati nel Machine Learning è proprio quello chiamato Random Forest. Esso è un algoritmo di tipo supervisionato che può essere implementato sia per task di classificazione che di regressione. La caratteristica di questo algoritmo è quella di essere un modello “ensemble⁸”, in particolare la tipologia di ensemble rappresentata dal random forest è quella denominata “bagging⁹”.

Funzionamento

Sempre D.Nardini [7], in un altro suo articolo analizza il funzionamento di tale algoritmo, mostrando aspetti molto interessanti al riguardo. Quando si decide di utilizzare un Random

⁸Modello che integra al suo interno altri modelli più semplici

⁹Tecnica di machine learning che fa parte dell’ensemble learning. Nel bagging, più modelli dello stesso tipo vengono addestrati su diversi set di dati, ognuno di essi viene ottenuto mediante campionamento casuale e permutazione (bootstrapping) dal set di dati originale.

Forest l’obiettivo sarà quello di creare molti classificatori deboli che uniti insieme possono portare ad un ottimo risultato. La caratteristica di questo algoritmo è che gli alberi sono indipendenti, cioè non si condizionano a vicenda: ciò permette di evitare alcuni degli errori più comuni degli alberi, infatti rischiano di essere molto instabili a causa del loro modo intrinseco di fare previsioni, cioè il metodo verticale ad albero. Altra importante caratteristica è quella di scegliere ogni volta un campione casuale di variabili da utilizzare per il train interno, il che rende ancora più indipendenti e diversi gli alberi, consentendo a ciascuno di essi di specializzarsi in modo diverso. Inoltre il bagging implica che alcuni dati possono comparire contemporaneamente in più modelli mentre altri potrebbero non comparire mai. Ogni albero vale per uno, e nessuno pesa più degli altri e questo è un punto di forza, ma bisogna comunque arrivare ad una conclusione. Possiamo immaginare il Random Forest come un parlamento: il modo di decidere è quello del voto, nel caso della classificazione.

Nella Figura 2.8 viene chiarito meglio questo concetto, infatti possiamo vedere una serie di N alberi decisionali, costruiti secondo la metodologia illustrata dalla Figura 2.7. Gli elementi del dataset che si vogliono classificare sono etichettati con due classi (classe A e classe B). Ognuno degli N alberi che costituiscono la foresta produce un risultato, e secondo il sistema di classificazione di tipo maggioritario la classe che otterrà più voti sarà quella con cui si contrasseggerà l’elemento in questione. Nel caso della nostra figura (osservando la struttura dei singoli alberi, e notando le foglie a quali classi sono associate) la probabilità di predizione per un elemento m del nostro dataset è del 66% per la classe A, e del 34% per la classe B.

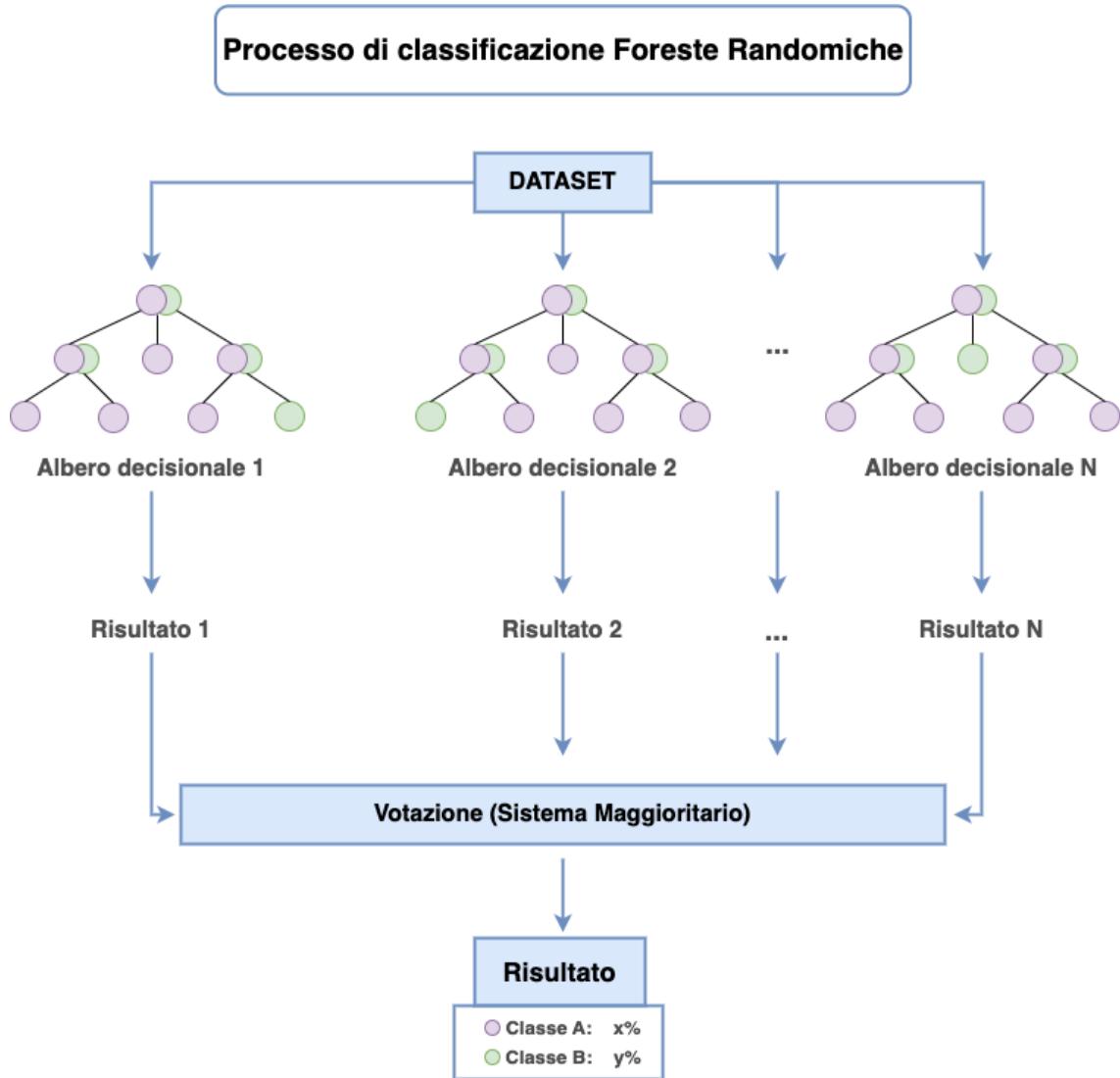


Figura 2.8: Processo di classificazione di un campione tramite algoritmo RF

Parametri del modello

Anche per il modello Random Forest si considerano fondamentali gli iper-parametri **criterion** e **max-depth**. Un altro parametro considerato è **min-samples-split**, ovvero il numero minimo di campioni necessari per dividere un nodo intero. In questo caso sarà un valore intero. Altro iper-parametro importante è **n-estimators** che ci dà la possibilità di scegliere il numero di alberi che faranno parte della foresta.

Pro e contro

PRO:

- Modello estremamente performante;

- Poca probabilità di incorrere in over-fitting;
- Utile sia per classificazione che per regressione;
- Abbastanza facile da interpretare, essendo formato da algoritmi molto semplici.

CONTRO:

- Tempo di esecuzione (solo se rapportato all'albero decisionale, stiamo parlando comunque di una manciata di secondi).

2.1.9 Macchina a vettori di supporto (SVM)

La Support Vector Machines (SVM) è un insieme di metodi di apprendimento supervisionato che possono essere utilizzati sia per fare classificazione sia per fare regressione.

Funzionamento

Il funzionamento della SVM viene trattato in modo chiaro nella documentazione di Scikit-learn¹⁰ [8]. In essa si legge che l'obiettivo dell'algoritmo della macchina vettoriale di supporto è trovare un iperpiano in uno spazio N-dimensionale che classifichi distintamente i punti dati. Per separare le N classi di punti dati ci sono molti possibili iper-piani che potrebbero essere scelti, l'obiettivo è trovare un piano che abbia il margine massimo, cioè la distanza tra i punti delle classi successive. La massimizzazione della distanza del margine fornisce un rafforzamento in modo che i punti dati futuri possano essere classificati con maggiore sicurezza, i vettori di supporto, invece, sono punti dati più vicini all'iperpiano e influenzano la sua posizione e il suo orientamento. L'uso di questi vettori di supporto massimizza il margine del classificatore, la rimozione dei vettori di supporto cambierà la posizione dell'iperpiano: questi sono i punti aiutano a costruire il SVM.

In Figura 2.9 viene mostrato graficamente il funzionamento di una SVM. Abbiamo la rappresentazione grafica di una serie di campioni appartenenti a due classi differenti (classe A e classe B). Tra questi vi sono 5 campioni (2 per la classe A e 3 per la classe B) che fungono da vettori di supporto per determinare il miglior iperpiano possibile (rettangolo blu) che permette una distinzione netta tra i campioni delle due classi.

¹⁰Libreria open source di apprendimento automatico per il linguaggio di programmazione Python <https://scikit-learn.org/stable/>

Processo di classificazione Alberi Decisionali

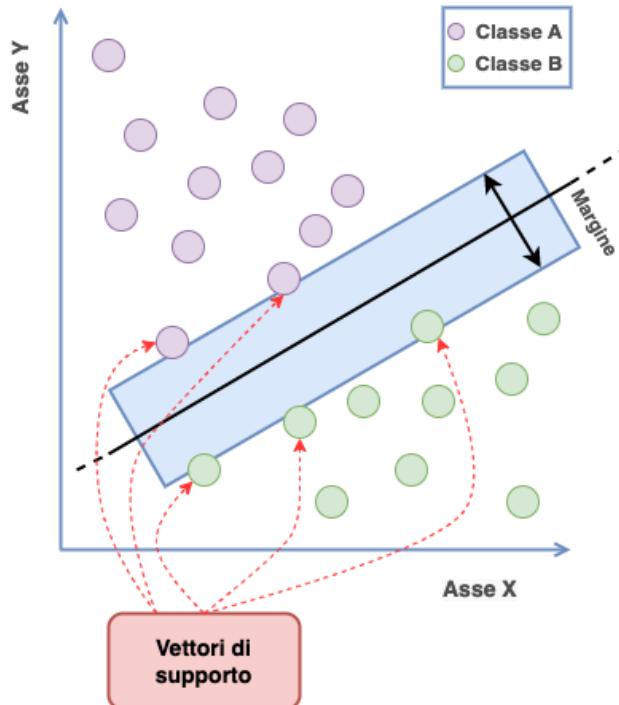


Figura 2.9: Processo di classificazione attraverso la Macchina a Vettori di Supporto

Parametri del modello

Principalmente si considera la **funzione del kernel** che è un metodo utilizzato per prendere i dati come input e trasformarli nella forma richiesta per elaborarli. Generalmente, trasforma l'insieme dei dati di addestramento in modo che una superficie decisionale non lineare sia in grado di trasformarsi in un'equazione lineare. In altre parole, restituisce il prodotto interno tra due punti in una dimensione caratteristica standard. Tali funzioni possono essere di diverso tipo:

- **Funzione di base radiale del kernel gaussiana (RBF):** viene utilizzata per eseguire la trasformazione quando non si hanno conoscenze preliminari sui dati;
- **Sigmoid Kernel:** questa funzione equivale ad un modello perceptron a due strati della rete neurale, che viene utilizzato come funzione di attivazione per i neuroni artificiali;
- **Kernel Polinomiale:** rappresenta la somiglianza dei vettori nell'insieme dei dati di addestramento in uno spazio di funzionalità rispetto ai polinomi delle variabili originali

utilizzate nel kernel.

Altro iper-parametro importante è il **valore C** che compensa la corretta classificazione degli esempi di addestramento con la massimizzazione del margine della funzione di decisione. Per i valori maggiori di C sarà accettato un margine inferiore, nel caso in cui la funzione di decisione è migliore nel classificare correttamente tutti i punti di allenamento; invece un valore più basso di C incoraggerà un margine più ampio, quindi una funzione decisionale più semplice, a scapito dell'accuratezza sul set di allenamento. In altre parole C si comporta come un parametro di regolarizzazione nella SVM.

Infine un altro parametro molto importante da tenere in considerazione è **gamma**: esso definisce fino a che punto arriva l'influenza di un singolo esempio di addestramento. Valori bassi significano “lontano” e valori alti invece significano “vicino”. Questo parametro è associato a tutte le diverse funzioni del kernel tranne quella lineare.

Pro e contro

PRO:

- Efficace in spazi ad alta dimensione;
- Utilizza un sottoinsieme di punti di addestramento nella funzione decisionale, quindi è anche efficiente in termini di memoria;
- È versatile, in quanto è possibile specificare diverse funzioni del kernel per la funzione di decisione.

CONTRO:

- Le SVM non forniscono direttamente stime di probabilità, queste sono calcolate utilizzando 5 costose convalide incrociate.

2.1.10 Rete Neurale Artificiale (ANN)

La Rete Neurale Artificiale è un sottoinsieme del machine-learning e costituisce il nucleo degli algoritmi di deep-learning. Il suo nome e la sua struttura sono ispirati al cervello umano e imitano il modo in cui i neuroni biologici si scambiano segnali.

Neurone artificiale

Un neurone artificiale è una funzione matematica che riceve un input, elabora quell'informazione e restituisce un output. La Figura 2.10 mostra un neurone con 3 input (X_1, X_2, X_3), ad ogni input è associato un peso (w_1, w_2, w_3), gli input con i relativi pesi vengono rangati nel neurone che poi restituirà l'output Y .

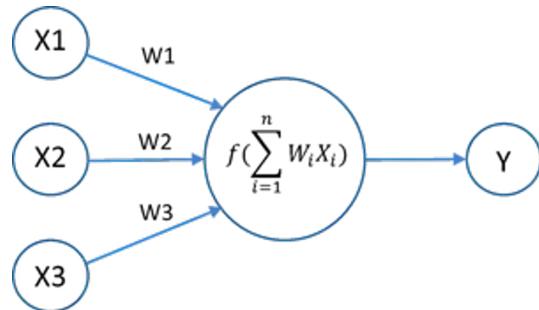


Figura 2.10: Struttura del neurone

Percettrone multi-nastro

Il percettrone multi-nastro è un architettura di rete neurale, costituita da una rete di neuroni (chiamati anche nodi) distribuiti su più layers:

- **Un layer di input:** in cui il numero di neuroni corrisponde al numero di proprietà del nostro datate;
- **Un layer di output:** in cui il numero di neuroni corrisponde al numero di classi;
- **Uno o più hidden layers:** livelli intermedi che utilizzano l'output del layer precedente per apprendere nuove proprietà.

Una rete neurale con un singolo hidden layer è anche definita "vanilla neural network" (VNN), mentre una rete con due o più hidden layers è anche definita deep neural network (rete neurale profonda o DNN). In Figura 2.11 viene mostrata la struttura di una vanilla neural network con 3 neuroni di input, un unico strato intermedio composto da una serie di neuroni e un neuroni di output. Mentre in Figura 2.12 si può vedere la struttura di una

rete neurale profonda con più di uno strato intermedio precisamente 3, un input layer con 3 neuroni e un output layer composto sempre da un neurone. Ogni neurone presente in queste due reti ha la struttura e il funzionamento mostrato precedentemente in Figura 2.10.

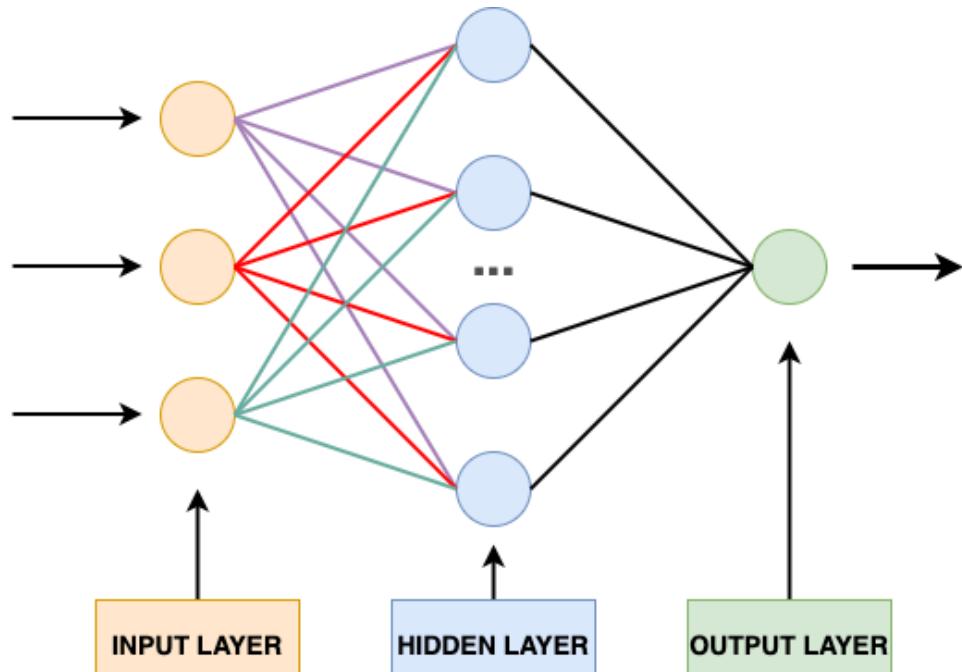


Figura 2.11: Struttura di una VNN

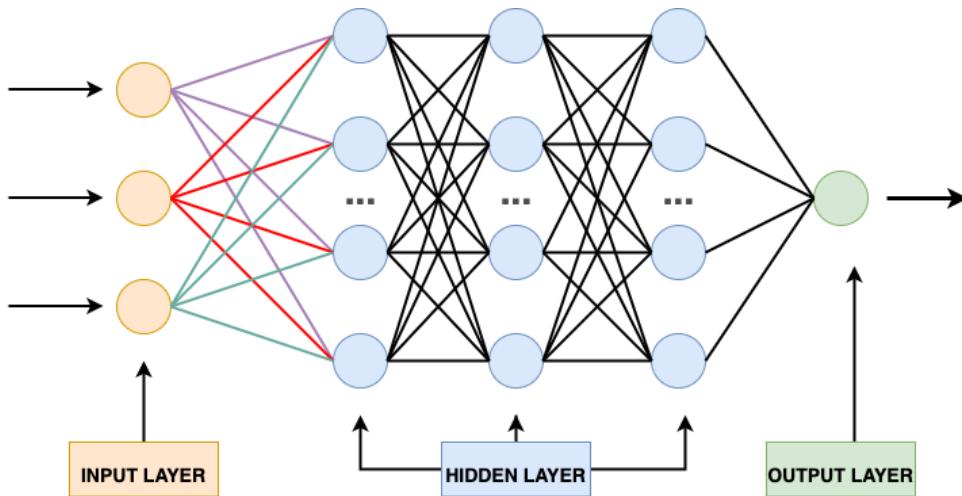


Figura 2.12: Struttura di una DNN

Funzionamento

Sempre nella documentazione di scikit-learn [8] si legge che ad ogni neurone di uno strato nascosto corrisponde un percepitrone e l'attivazione di ciascuno di essi è data da una funzione di attivazione non lineare. Ogni nodo o neurone artificiale è connesso a un altro nodo e ha

pesi e soglie associati. Quando l'output di un singolo nodo supera una soglia specifica, quel nodo si riattiva e invia i dati al livello di rete successivo; in caso contrario i dati non verranno trasferiti al livello successivo della rete.

Parametri del modello

Ci sono diversi iper-parametri che caratterizzano una ANN. Il principale è sicuramente la **funzione di attivazione** per il livello nascosto che determina se attivare o meno il neurone, ciò significa utilizzare semplici operazioni matematiche per determinare se l'input di un neurone nella rete è rilevante per il processo di previsione. Lo scopo di una funzione di attivazione è quello di introdurre una non linearità in una rete neurale artificiale, producendo un output da una raccolta di valori di input dati allo strato. Principalmente si considerano due tipologie:

1. **Relu**(unità lineare rettificata): una funzione facile da calcolare, e dato che tutti i neuroni non sono attivati, questo crea scarsità nella rete e quindi sarà veloce ed efficiente. Tuttavia tale funzione non è centrata su zero e può uccidere alcuni neuroni per sempre poiché da sempre 0 per valori negativi;
2. **Tanh** (tangente iperbolica): una funzione centrata su zero ed ha un intervallo compreso tra $(-1, 1)$ e questi ne caratterizzano i suoi vantaggi, anche se è computazionalmente costosa.

Poi vi è il risolutore per l'ottimizzazione del peso, questo cambia a seconda della tipologia di campioni che contiene il dataset, ad esempio: "**adam**" funziona abbastanza bene su set di dati relativamente grandi, nel caso di questo lavoro MNIST e F-MNIST.

Associato al peso vi è un altro parametro che è il **learning-rate**, ossia il programma del tasso di apprendimento per gli aggiornamenti del peso. Principalmente si considerano due tipologie ossia:

1. **constant**: rappresenta una velocità di apprendimento costante;
2. **adaptive**: mantiene la velocità di apprendimento costante finché la perdita di allenamento continua a diminuire.

Poi si considera **imax-iter**, che nel caso in cui il risolutore è Adam rappresenta il **numero di epoch**, ossia quante volte verrà utilizzato ciascun punto dati. Ultimo è il valore **hidden-layer-sizes** che ci permette di settare il numero di hidden layers che costituirà la rete neurale e la quantità di neuroni nascosti per ogni livello. Avere più livelli intermedi e un gran numero

di neuroni permette al modello di separare i dati in modo non lineare. Quindi maggiori saranno i neuroni, maggiore sarà la complessità che tale rete può gestire.

Pro e contro

PRO:

- Capacità di classificare pattern complessi e non lineari come le immagini, video, suoni e testi;
- Capacità di lavorare in parallelo;
- Tolleranza a agli errori e al rumore;
- Alta precisione;
- Capacità di generalizzazione.

CONTRO:

- Servono per risolvere problemi specifici;
- Occorrono dataset di una certa dimensione;
- Possono richiedere tempi di training molto lunghi.

2.2 Stato dell’Arte

Le nozioni mostrate nella sezione di Background precedente permettono di comprendere meglio quelli che sono stati i lavori fatti in letteratura per quanto riguarda la classificazione delle immagini mediante tecniche e algoritmi di Intelligenza Artificiale.

In letteratura molti hanno preso in considerazione come oggetto di studio gli algoritmi di intelligenza artificiale testando le loro prestazioni in compiti di classificazione. Tra i tanti dataset utilizzati per addestrare tali algoritmi ad effettuare classificazioni di campioni, quelli maggiormente usati sono stati MNIST e F-MNIST che, come abbiamo visto, sono i migliori nell’ambito della classificazione delle immagini. Tuttavia, lo studio di tali algoritmi è mirato più che a comprendere la loro vera efficienza quanto a combinare i loro iper-parametri al fine di ottenere un’accuratezza sempre maggiore.

R. Cervelli [9] asserisce che le applicazioni basate sul ML non intendono sostituire le applicazioni tradizionali che eseguono operazioni già programmate in modo più rapido e accurato, ma poiché apprendono direttamente dalle esperienze dei dati possono risolvere nuovi problemi, affrontare scenari complessi e utilizzare l’intuizione e la memoria per comportarsi come esseri umani. Ciononostante, il rischio che un algoritmo passi dall’apprendere all’imparare sistematicamente è molto alto. Per questo motivo la scelta degli iperparametri diventa un lavoro minuzioso e la loro combinazione può portare a miglioramenti o alla completa inutilità del modello.

2.2.1 Riconoscimento delle cifre scritte a mano

Nel 2017 Hamid e Sjarif [10] hanno effettuato un lavoro di comparazione tra diversi classificatori quali KNN, SVM e percepitrone Multi-nastro per la classificazione di cifre scritte a mano del MNIST. Lo studio su questi classificatori si è sviluppato in due fasi: nella prima fase sono stati scelti il pool di iper-parametri migliore per ogni classificatore, mentre nella seconda fase i classificatori sono stati addestrati con i campioni di cifre a mano messe a disposizione dal dataset MNIST, per poi infine essere testati su un campione di cifre da loro fornito, per SVM e MPL, mentre per KNN da un pool random estratto dal MNIST. In Figura 2.13 viene mostrato l’input di cifre scritte a mano fornito ai classificatori da Hamid e Sjarif. Come si può vedere è una sequenza di 8 cifre di cui ognuna di esse appartiene ad una delle 10 classi con cui sono organizzati i campioni nel MNIST.

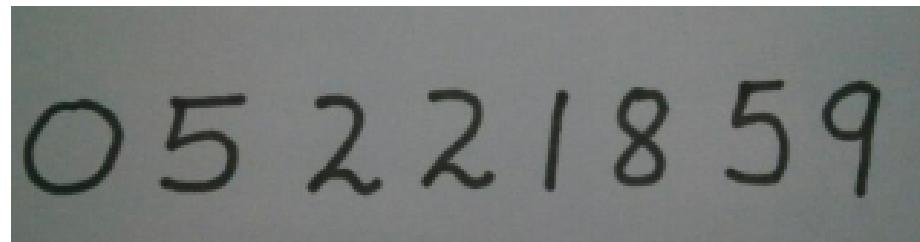


Figura 2.13: Campione di 8 cifre scritte a mano utilizzato per testare i classificatori KNN, SVM e MPL

Per quanto riguarda il classificatore KNN, i due si sono concentrati sulla scelta del numero di vicini (in un range compreso tra 1 e 30) come iper-parametro cardine del modello. Come si può vedere in Tabella 2.2, i risultati ottenuti mostrano la miglior percentuale di accuratezza del classificatore: 99,26% con numero di vicini compreso tra 1 e 15.

Vicini	Accuratezza
1	99,26%
3	99,26%
5	99,26%
7	99,26%
9	99,26%
11	99,26%
13	99,26%
15	99,26%
17	98,52%
19	98,52%
21	97,78%
23	97,04%
25	97,78%
27	97,04%
29	97,04%

Tabella 2.2: Risultati esperimento di classificazione di cifre scritte a mano con KNN

Per la SVM e MPL invece è stato effettuato un lavoro di pre-elaborazione dell’immagine caratterizzato da due step:

1. Conversione dell’immagine in scala di grigi;

2. Applicazione del filtro gaussiano;

Dopodiché sono stati individuati i contorni limite dell’immagine sul quale poi realizzare i rettangoli di individuazione della cifra sovrapposti ai contorni individuati. I risultati da loro ottenuti sono mostrati in Figura 2.14a per quanto riguarda SVM (Tutti i campioni classificati correttamente), e in Figura 2.14b per quanto riguarda MPL (tutti i campioni classificati correttamente fatta eccezione per la cifra 9 riconosciuta come 8).

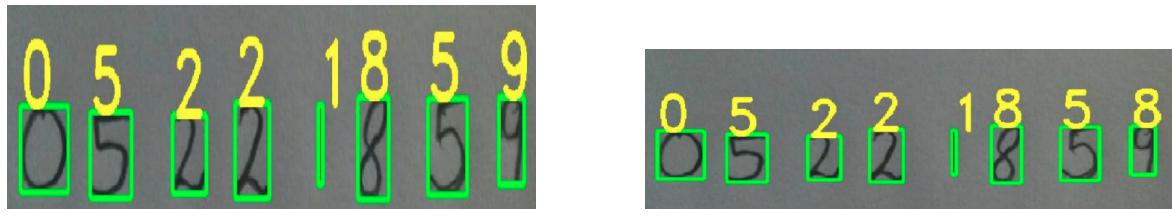


Figura 2.14: Risultati di classificazione tramite SVM e MPL messi a confronto

In conclusione Hamid e Sjarif hanno affermato che nonostante l’errore di classificazione per una cifra nella classificazione utilizzando il percepitrone multi-nastro, il settaggio degli iper-parametri ha fornito dei risultati più che soddisfacenti su tutti e tre i classificatori.

2.2.2 Sistema di riconoscimento ottico dei caratteri di cifre scritte a mano

Qualche anno prima nel 2013 El Kessab [11] con il suo team di sviluppo, hanno addestrato e testato un classificatore di rete neurale utilizzando la banca dati MNIST. Nel loro lavoro si può vedere come abbiano individuato una tecnica di apprendimento efficiente utilizzando l’algoritmo di discesa del gradiente. Nel processo di apprendimento viene modificato quello che è il peso sinaptico delle connessioni tra i neuroni. Per fare tutto ciò hanno proposto come classificatore un percepitrone multi-nastro per il riconoscimento delle immagini binarie (pixel bianchi e neri). Come si può ben vedere la fase di classificazione è preceduta da una fase di pre-elaborazione delle immagini e di estrazione di queste ultime dal dataset. Nella fase di preelaborazione hanno ridotto il rumore¹¹ delle immagini, e le hanno normalizzate estraendole con dimensioni 28x28.

Invece nella fase di estrazione il metodo da loro utilizzato ha previsto la divisione del campione in cinque zone caratteristiche: ovest, est, nord, sud e centrale. Queste zone caratteristiche sono state rilevate dalla dilatazione¹² delle immagini elaborate nelle quattro direzioni (quella centrale è esclusa).

¹¹Variazione casuale della luminosità o delle informazioni sul colore delle immagini.

¹²Trasformazione basata sulla forma dell’immagine.

Nella Figura 2.15 viene mostrata nel dettaglio l'estrazione delle caratteristiche delle 5 zone di un campione appartenente al dataset MNIST corrispondente alla cifra 5. Andando in ordine, partendo dall'alto, possiamo vedere l'estrazione delle caratteristiche della zona nord, successivamente quelle della zona ovest, centrale ed est, ed infine quelle della zona sud.



Figura 2.15: Schema di estrazione delle caratteristiche in base alla zona

Infine per classificare hanno utilizzato il percepitrone multi-nastro della famiglia delle reti neurali artificiali, il cui iper-parametro di studio è stato il numero di neuroni dello strato nascosto. Questo veniva scelto in base a tre condizioni:

1. Uguagliare il numero di neuroni dello strato di input;
2. Pari al 75% del numero di neuroni dello strato di ingresso;
3. Pari alla radice quadrata del prodotto dei due strati di uscita e di entrata. Abbiamo seguito queste tre condizioni; abbiamo variato il numero di neuroni nello strato nascosto tra quattro e nove neuroni. Il metodo utilizzato per l'apprendimento è l'algoritmo di back-propagation a gradiente: 20, 21, 22, 23 e 24.

Il tasso di riconoscimento ottenuto è stato dell’ 80,00% con un database di prova contenente 60.000 campioni. In Figura 2.16 viene mostrata la matrice di confusione, utile a comprendere il match tra le classi di riconoscimento. Gli indici delle colonne mostrano la classe predetta, mentre gli indici delle righe mostrano la classe corretta, il valore di accuratezza è pari al valore medio ottenuto tra quelli che compongono la diagonale principale della matrice.

Digits	0	1	2	3	4	5	6	7	8	9
0	86.45	00.81	01.10	00.01	01.40	00.27	00.22	00.17	06.36	03.21
1	00.02	94.39	01.00	00.84	00.02	00.48	00.44	00.18	01.03	01.59
2	00.09	01.42	88.73	04.33	00.82	00.65	00.03	01.00	02.00	00.93
3	00.06	00.52	00.96	77.02	01.18	00.45	00.00	14.53	04.95	00.32
4	01.21	02.92	00.34	01.44	77.94	02.85	03.30	00.58	03.59	05.83
5	00.03	01.37	01.27	01.03	00.74	84.10	06.73	00.22	02.58	01.92
6	02.29	02.41	00.81	00.10	06.21	02.43	78.81	00.00	06.78	00.15
7	00.04	03.47	06.66	03.83	01.50	00.04	00.00	77.12	04.84	02.48
8	04.82	02.58	01.50	02.03	04.08	00.22	00.70	03.16	79.03	01.86
9	04.85	09.89	08.34	00.65	02.70	08.14	02.51	01.90	11.37	49.64

Figura 2.16: Matrice di confusione dei risultati ottenuti dalla classificazione delle cifre scritte a mano con MPL

2.2.3 Classificazione F-MNIST basata sul descrittore delle caratteristiche HOG

Nel 2019, Greeshma e Sreekumar [12] hanno presentato la classificazione del dataset F-MNIST utilizzando il descrittore HOG (Istogramma del gradiente orientato) e la macchina a vettori di supporto (SVM) vista nella sezione di background precedente.

Secondo i ricercatori uno dei metodi più semplici ed efficaci di estrazione delle caratteristiche è il descrittore HOG, grazie anche alla sua semplicità nei calcoli. Con HOG infatti è possibile descrivere la forma e l’aspetto dell’immagine dividendola in piccole celle come (4 4) nel caso del loro lavoro, e calcola le direzioni dei bordi. Per migliorare l’accuratezza gli istogrammi possono essere normalizzati.

Nella Figura 2.17 vengono mostrate le caratteristiche HOG che sono state estratte da un’immagine appartenente al dataset F-MNIST utilizzando tre diverse dimensioni di cella (2 2), (4 4), (8 8). Da ciò si può evincere che le dimensioni della cella (2 2) contiene più informazioni relative alla forma rispetto alle dimensioni della cella (8 8), tuttavia la dimensionalità del vettore di caratteristiche utilizzando HOG aumenta rispetto al primo. Un ottimo compromesso

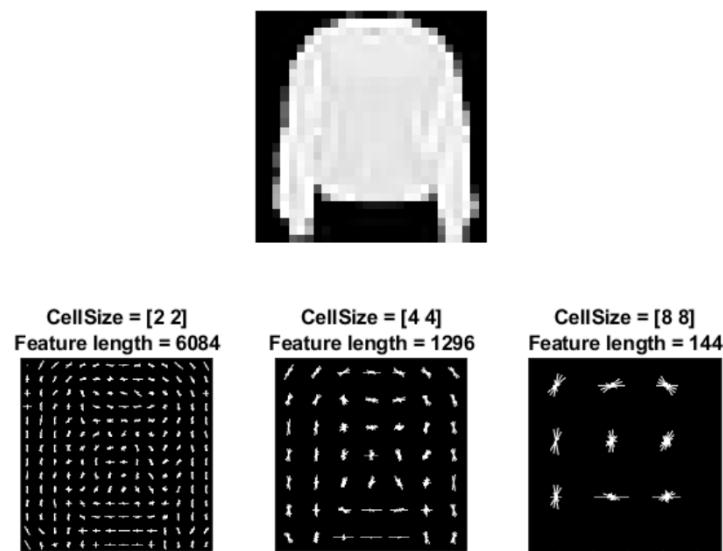


Figura 2.17: Caratteristiche estratte da un immagine campione di F-MNIST

sarebbe infatti la dimensione della cella (4 4) in quanto il numero di dimensioni è limitato a ciò che contribuisce a velocizzare il processo di addestramento, e contiene informazioni a sufficienza per visualizzare la forma dell’immagine di moda.

Il loro lavoro è stato implementato in MATLAB. Le caratteristiche HOG delle immagini vengono estratte dalle immagini di addestramento 28 x 28 pixel di F-MNIST utilizzando la funzione in MATLAB. Prima di tutto, nella fase di addestramento, si estraggono le caratteristiche HOG dalle immagini di addestramento e poi si utilizzano per fare previsioni con il classificatore. Le caratteristiche HOG estratte vengono utilizzate per addestrare il classificatore. I risultati vengono valutati utilizzando le immagini del set di prova e per misurare l’accuratezza del classificatore viene prodotta una matrice di confusione. La dimensione delle celle utilizzata per le caratteristiche HOG è (4 4). Le caratteristiche di 60000 immagini vengono poi inserite in SVM multi classe per l’addestramento. Infine, il test viene condotto su 10000 immagini del set di test. L’algoritmo raggiunge un’accuratezza dell’ 86,53% sulle immagini di prova.

La Figura 2.18 mostra la matrice di confusione delle classi nel dataset di immagini di moda, dove si capisce chiaramente che l’incertezza si è verificata tra le categorie ‘0’ e ‘6’, ‘4’ e ‘2’, ‘4’ e ‘6’, ‘2’ e ‘6’, che hanno senso perché t-shirt e camicie, cappotto e pullover e cappotto e camicie hanno lo stesso aspetto e sono un po’ confuse.

Class	Label	0	1	2	3	4	5	6	7	8	9
T-shirt/ top	0	835	2	14	26	4	1	109	0	9	0
Trouser	1	2	963	2	27	1	0	5	0	0	0
Pullover	2	15	1	765	9	113	0	92	0	5	0
Dress	3	26	9	13	871	35	0	43	0	3	0
Coat	4	2	1	95	37	796	0	68	0	1	0
Sandal	5	0	1	0	0	0	950	0	38	1	10
Shirt	6	148	0	84	35	110	0	612	0	11	0
Sneaker	7	0	0	0	0	0	31	0	952	0	17
Bag	8	2	1	5	7	6	3	10	1	965	0
Ankle boot	9	0	0	0	0	0	13	1	42	0	944

Figura 2.18: Matrice di confusione delle caratteristiche HOG su F-MNIST

2.2.4 Addestramento efficiente di alberi decisionali robusti con GROOT

Nel 2021 Vos e Verner [13] hanno presentato GROOT, un algoritmo efficiente per l’addestramento di alberi decisionali robusti e foreste casuali. Lo studio è stato eseguito su un iper-parametro in particolare, ossia il criterio con il quale i dati vengono spartiti, in particolar modo si sono concentrati sull’impurità di Gini. Mentre nei classici algoritmi utilizzati per addestrare gli alberi decisionali tale criterio viene calcolato iterativamente, Vos e Verner nella loro implementazione hanno proposto una risoluzione di questa funzione in modo analitico¹³.

¹³Approccio che fa da ponte tra le funzioni polinomiali e le funzioni generiche

Nel loro lavoro ipotizzano che vi sia l’esistenza di un attaccante che conosce il modello e perturba i campioni in modo da renderli difficilmente riconoscibili. Questo algoritmo da loro presentato funziona in questo modo:

- Itera su ogni campione in ordine sparso per identificare gli split candidati;
- Valuta su ogni split l’impurità di Gini avversaria per trovare lo split più accurato contro l’avversario.

Di seguito è mostrato lo pseudo-codice:

Algorithm 1 Trovare il miglior Split robusto per i valori delle caratteristiche

Input: valori delle caratteristiche X limiti di perturbazione (ϵ_l, ϵ_r)
 x_1 si riferisce ai campioni con label 1
 $S \leftarrow X\{o - \epsilon_r | o \in X\} \cup \{o + \epsilon_r | o \in X\}$
for $s \in S$ **do**
 $R \leftarrow \{o | o \in X \wedge o > s + \epsilon_r\}$
 $RI \leftarrow \{o | o \in X \wedge s < o \leq s + \epsilon_r\}$
 $LI \leftarrow \{o | o \in X \wedge s - \epsilon_l < o \leq s\}$
 $L \leftarrow \{o | o \in X \wedge o \leq s - \epsilon_l\}$
 $(m_{1s}, m_{0s}) \leftarrow$ numero di campioni da I da muovere verso sinistra
 $m_{1s} \leftarrow round(m_{1s}), m_{0s} \leftarrow round(m_{0s})$
 $g_s \leftarrow S(|L_0| + m_{0s}, |L_1| + m_{1s}, |R_0| + |I_0| -$
 $m_{0s}, |R_1| + |I_1| - m_{1s})$
end for
 $s' \leftarrow argmin_s g_s$
split con soglia s'
Output $(s', g'_s, m_{1s'}, Y'_s)$

Infine, hanno testato l’algoritmo su i set di immagini MNIST e F-MNIST, essendo però GROOT limitato ai problemi di classificazione binaria i set di dati sono stati modificati nel seguente modo:

- MNIST -> MNIST 2 vs 6;
- F-MNIST -> F-MNIST sandali vs sneakers.

I risultati ottenuti hanno confermato una velocità di esecuzione dell’algoritmo maggiore rispetto a quelli tradizionali, e allo stesso tempo una percentuale di accuratezza molto alta.

In Figura 2.19a vengono mostrate le perturbazioni minime per far sbagliare il classificatore a catalogare campioni di cifra "2" e "6" del set di dati MNIST. Mentre la Figura 2.19b lo fa per i campioni "sandali" e "sneakers". Le matrici delle immagini generate presentano due esempi per ogni avversario e ogni colonna mostra l'output necessario per indurre in errore il classificatore, come si può osservare l'algoritmo GROOT è quello con le immagini più disturbate delle altre, sintomo di un buon range di affidabilità nella classificazione anche quando le immagini sono poco chiare.

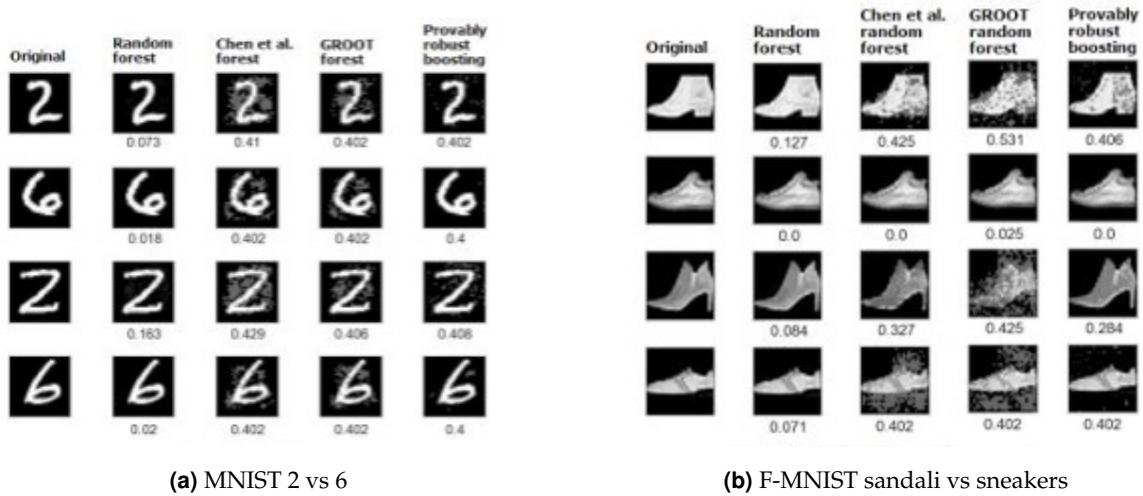


Figura 2.19: Risultati di classificazione messi a confronto

In Tabella 2.3 invece, viene mostrata la trascrizione dei loro risultati ottenuti per quanto riguarda l'accuratezza della classificazione su MNIST e su F-MNIST. Come si può vedere GROOT per MNIST si posiziona come secondo per quanto riguarda l'accuratezza sul set di test, mentre su F-MNIST si posiziona come terzo.

MNIST 2 vs 6			
Modello	Accuratezza	Accuratezza avversaria	Tempo
Random Forest	99,7%	0,0%	3.9 sec
Chen et all Forest	98,9%	89,1%	2.1 min
GROOT Forest	99,4%	91,9%	2.5 min
boosting dimostrabilmente robusto	99,2%	92,7%	5.1 hr

F-MNIST sandali vs sneakers			
Modello	Accuratezza	Accuratezza avversaria	Tempo
Random Forest	95,8%	0,0%	5,9 sec
Chen et all Forest	90,0%	52,6%	4.3 min

GROOT Forest	89,0%	70,4%	5.0 min
boosting dimostrabilmente robusto	88,9%	75,9%	13,5 ore

Tabella 2.3: Risultati di accuratezza sul set di test di MNIST e F-MNIST messi a confronto

2.2.5 Limiti sullo Stato dell’Arte

Come si può vedere, in letteratura sono presenti molti lavori che analizzano gli algoritmi di machine learning. Questi vengono studiati attentamente, ottimizzati, modificati, sempre al fine di ottenere un unico risultato, ossia l’accuratezza nella classificazione. Tuttavia, questo studio molto spesso si sofferma su una singola caratteristica di questi algoritmi e oltretutto non sempre vi è un confronto dettagliato sulle performance di questi ultimi su vari dataset.

Riprendendo un concetto precedentemente citato, al giorno d’oggi stiamo assistendo ad un vero e proprio passaggio di testimone tra il dataset MNIST e F-MNIST, destinato a cambiare nuovamente gli standard di accuratezza degli algoritmi di intelligenza artificiale. Questo lavoro di tesi è volto ad analizzare ancor più nel dettaglio quelli che sono gli algoritmi di ML mostrati precedentemente, individuando i loro migliori iper-parametri e combinarli al fine di mostrare la loro reale accuratezza; mettendo a confronto le performance che hanno rispettivamente con MNIST e F-MNIST giustificando questo cambio.

CAPITOLO 3

Metodologia di ricerca

Nel seguente capitolo è presentato l'obiettivo della ricerca e sono descritti i materiali e i metodi utilizzati per condurre tale studio.

3.1 Obiettivo della ricerca

La presente ricerca ha l'obiettivo di individuare i migliori iper-parametri che incidono sull'accuratezza degli algoritmi di Machine Learning presentati nel capitolo 2. Per ogni algoritmo vengono presentate una serie di combinazioni di iper-parametri, al fine di scegliere quella che garantisce la miglior prestazione nel classificare campioni di immagini appartenenti ai set di dati MNIST e F-MNIST. I risultati ottenuti in termini di accuratezza vengono poi confrontati al fine di decidere quali dei due dataset è effettivamente "più difficile" e di conseguenza più affidabile.

3.2 Materiali

Ai fin della ricerca sono stati considerati gli algoritmi di ML forniti dalla libreria, open source, scikit-learn. La scelta degli iper parametri è avvenuta consultando la documentazione specifica di ogni classificatore:

- K-Nearest Neighbors [14] [15];
- Alberi Decisionali [14] [16];

- Random Forest [14] [17];
- Macchina a vettori di supporto [14] [18];
- Reti neurali artificiali (percettrone multi-nastro) [14] [19].

3.3 Metodi

Il lavoro effettuato può essere diviso in tre fasi:

1. Svolgimento di operazioni preliminari sui dataset MNIST e F-MNIST al fine di prepararli per addestrare i modelli considerati.
2. Scelta dei migliori iper-parametri tra quelli disponibili, che influenzano l'accuratezza di classificazione del modello;
3. Simulazione dei modelli con tutte le combinazioni possibili di iper-parametri scelti, al fine di trovare le 10 migliori combinazioni, e successivo addestramento, effettivo, dei modelli per ottenere i risultati di accuratezza;

3.3.1 Operazioni preliminari alla classificazione

Suddivisione dei dati presenti in MINST e F-MNIST

Per poter effettuare la classificazione in modo ottimale, gli algoritmi di ML devono prima essere addestrati alla classificazione. Per questo motivo i dati presenti in MNIST e F-MNIST sono stati suddivisi in due set più piccoli: uno per il train del modello e uno per il test del modello. I set di dati sono stati organizzati in 4 array NumPy:

- **X-train e Y-train:** Array bidimensionali che contengono per ogni riga tutte le caratteristiche di un campione in termini di pixel;
- **X-test e Y-test:** Array monodimensionali che contengono per ogni posizione un valore compreso tra (0-9). Come abbiamo visto nel Capitolo 2 ogni valore corrisponde ad un'etichetta di classificazione.

Una riga i-esima dell'array bidimensionale X-train/Y-train e un valore nella i-esima posizione dell'array monodimensionale X-test/Y-test corrisponde ad un record completo del dataset.

La scelta effettuata ha previsto che su 70.000 campioni, 60.000 venissero utilizzati per il train e 10.000 per il test in modo da avere un pool equo tra tutte le classi del dataset (10 classi, circa 1000 campioni per classe). Utilizzare l'85% dei dati per addestrare il modello e

il restante 15% per il test è una scelta dovuta a due fattori principali. Il primo è l'estrema difficoltà dei dataset, ogni campione è caratterizzato da molte proprietà (una su ogni pixel che compone l'immagine del campione). Il secondo fattore è l'elevato numero di classi con le quali i campioni si possono classificare (ben 10 labels). Per questo motivo, i modelli hanno bisogno di molti campioni di addestramento per poter apprendere.

In Figura 3.1 viene mostrato un grafico a torta per la ripartizione dei dati di MNIST e F-MNIST nei due sotto-set.

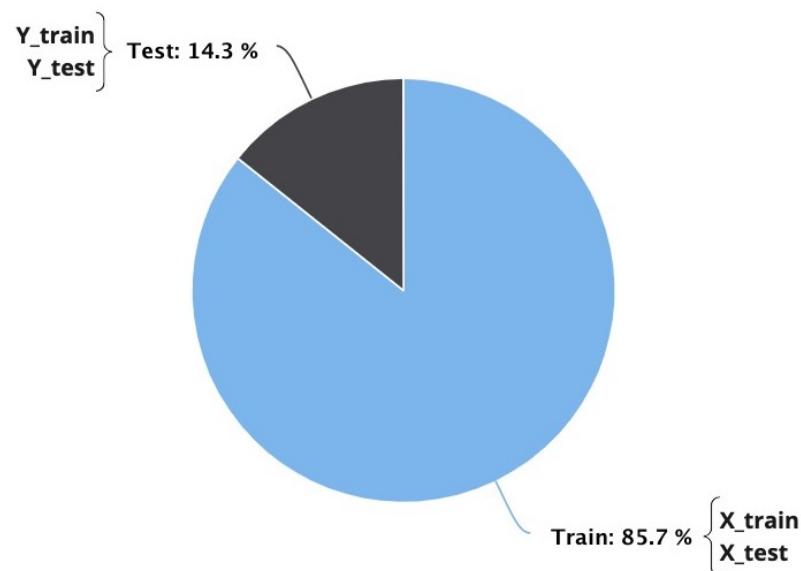


Figura 3.1: Grafico a torta ripartizione dataset MNIST/F-MNIST nei due set di train e test

Standardizzazione del dataset

Altra operazione molto importante prima dell'approccio all'addestramento del modello è stata la normalizzazione delle caratteristiche in un determinato intervallo, in modo tale da avere tutte le proprietà di ogni singolo campione ridimensionate alla stessa scala. Questo è stato possibile grazie alla funzione "**MinMaxScaler(...)**" offerta dalla libreria scikit-learn. Questo è uno stimatore che ridimensiona e traduce ogni caratteristica individualmente in modo tale che sia nell'intervallo dato sul set di addestramento. Essendo MNIST e F-MNIST, dei dataset contenenti immagini come campioni questi sono in scala [1, 256], pertanto sono

stati trasformati in scala $[0, 1]$ secondo la seguente formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Ciò significa che i valori minimo e massimo di una caratteristica/variabile sono rispettivamente 0 e 1.

3.3.2 Scelta degli iper-parametri

La scelta degli iper-parametri è stato un lavoro suddiviso in 3 step:

1. Consultazione della documentazione ufficiale della libreria scikit-learn per individuare gli iper-parametri da considerare;
2. Scelta dell' iper-parametro principale (quello che tra tutti influenza di più l'accuratezza del classificatore)
3. Definire per ogni iper-parametro i valori che esso può assumere.

K-Nearest Neighbors

Per questo algoritmo l' iper-parametro principale scelto è stato il numero di vicini (**valore k**). Per esso si sono scelti una serie di valori compresi nel range $(1, 21)$. Gli iper-parametri secondari scelti sono stati invece:

- **Metrica di distanza:** euclidean, manhattan e minkowski;
- **I pesi:** uniforme e distanza.

Il classificatore è stato addestrato aumentando in modo crescente il numero di vicini per comprendere come variava la sua accuratezza nella classificazione. Per questo algoritmo si è scelto di effettuare un addestramento per numero crescente di vicini una volta per ogni metrica di distanza e per peso, per un totale di 126 combinazioni.

Alberi Decisionali

Per questo algoritmo l' iper-parametro principale scelto è stata la profondità massima dell'albero(**max-depth**). Per esso si sono scelti una serie di valori compresi nel range $(1, 21)$. Gli iper-parametri secondari scelti sono stati invece:

- **Criterio di split:** Gini e Entropia;

- **Numero minimo di campioni che possono finire nelle foglie:** con valore crescente (1, 25, 50);
- **Numero di variabili da utilizzare per trovare il miglior split:** Si è scelto di utilizzare il valore 0,5 (lo split verrà cercato sulla radice quadrata del numero di variabili) e poi None (lo split verrà cercato su tutte le variabili).

Il classificatore è stato addestrato aumentando in modo crescente la sua profondità per comprendere come variava la sua accuratezza nella classificazione. Per questo algoritmo si è scelto di effettuare un addestramento per profondità massima crescente una volta per ogni criterio di split, per numero minimo di campioni nelle foglie e per numero minimo di variabili per trovare il best split per un totale di 252 combinazioni.

Random Forest

Essendo le Foreste randomiche un insieme di alberi decisionali, in questo caso gli iper-parametri principali scelti sono stati due:

- **Numero di alberi che compongono la foresta (n-estimators):** valori compresi nel range (100, 200);
- **Massima profondità di ogni albero (max-depth):** serie di valori compresi nel range (1, 21).

Gli iper-parametri secondari scelti sono stati invece:

- **Criterio di split:** Gini e Entropia;
- **Numero minimo di campioni per dividere un nodo intero (min-samples-split):** con valore intero crescente (2, 5, 10).

Il classificatore è stato addestrato aumentando in modo crescente il numero di esemplari che costituiscono la foresta simultaneamente all'aumento della profondità massima per ogni albero. Tali valori sono stati poi combinati una volta per ogni criterio e per numero minimo di campioni utili a dividere un nodo intero. In totale per questo algoritmo sono state provate 36 combinazioni.

Macchina a Vettori di Supporto

Per questo algoritmo il lavoro effettuato è stato leggermente diverso. L'accuratezza di questo modello non ha un vero e proprio iper-parametro principale. La combinazione di **kernel**,

valore C e Gamma può far ottenere al classificatore degli eccellenti risultati o risultati pessimi anche alla minima variazione di un singolo parametro dei tre. Inoltre essendo modelli di classificazione molto robusti hanno un elevato tempo di addestramento, per questo motivo si è deciso di optare soltanto per una simulazione di addestramento, leggermente più imprecisa, ma meno costosa in termini di tempo. Le funzioni kernel prese in considerazione sono state: "rbf", "sigmoid" (la più costosa in termini di tempo di addestramento) e "polinomiale". Per ognuna di esse è stato associato un valore c (una volta 5 e una volta 10), e il parametro gamma con valore decrescente (0,01, 0,001, 0,0001) per un totale di 18 combinazioni di simulazione.

Reti Neurali artificiali (Percettrone Multi-nastro)

Per questo classificatore gli iper-parametri principali individuati sono due:

- **Numero di hidden layers:** crescente da 1 fino ad una rete con 4 livelli intermedi;
- **Numero di neuroni per livello:** In questo il valore scelto è stato correlazione alla complessità del dataset. Essendo le immagini campione di MNIST meno dettagliate rispetto a quelle di F-MNIST, per il primo si è deciso di considerare un numero di neuroni pari a 100. Per il secondo invece si è considerato un numero di neuroni molto più elevato: prima 250 per ogni livello e poi 512.

Gli iper-parametri secondari scelti sono stati invece:

- **la funzione di attivazione:** Tahn e Relu (le principali per decidere quali neuroni devono attivarsi durante la classificazione di ogni campione);
- **La velocità di apprendimento:** sia costante che adattiva;
- **Il risolutore:** che nel caso della classificazione di immagini l'unico possibile che offre ottime performance è **Adam**;
- **il numero di epoch o cicli :** utilizzando il valore di default fornito dalla libreria che è 200, considerato un valore consueto all'addestramento e alla classificazione, considerando il fatto che per la maggior parte dei dataset le iterazioni non superano mai quella soglia anzi sfermano prima.

Il classificatore per MNIST è stato addestrato aumentando in modo crescente il numero di livelli intermedi una volta per ogni funzione di attivazione e per velocità di apprendimento, per un totale di 32 combinazioni. Per F-MNIST le combinazioni sono il doppio ossia 64, in quanto si è tenuto conto anche del numero di neuroni per ogni livello (due valori anziché uno).

3.3.3 Simulazione e addestramento effettivo

Al fine di simulare e poi addestrare i modelli per la classificazione dei campioni (raffinati nelle fasi precedenti), sono stati realizzati due tipologie di script Python per ogni classificatore.

- **Script tipologia 1:** Simula l’addestramento del classificatore su un dizionario di iper-parametri fornito come input. Ad ogni iterazione il classificatore è stato simulato con una combinazione di iper-parametri differente. Una volta terminate le simulazioni con tutte le possibili combinazioni di iper-parametri, è stato fornito in output:
 - Le 10 migliori combinazioni che hanno permesso al modello di ottenere i valori migliori in termini di accuratezza nella classificazione, sia sul set di train, che sul set di test, ma anche il valore più basso sul tasso di errore;
 - Il probabile valore attorno al quale si aggirerà l’accuratezza massima del classificatore. Questo perché lo script ha simulato il classificatore ma non lo ha testo effettivamente;
 - Un grafico in formato ".png" che mostra l’accuratezza sul set di test e sul set di train al variare dell’iper-parametro che più incide sul modello.
- **Script tipologia 2:** Testa l’addestramento del classificatore con una serie di iper-parametri forniti in input. La serie di iper parametri è stata scelta in base ai risultati ottenuti dall’esecuzione del primo script. In output è stato restituito:
 - L’accuratezza sul set di train;
 - L’accuratezza sul set di test;
 - Il tasso di errore di classificazione sul set di train;
 - Il tasso di errore di classificazione sul set di test;
 - Il report di classificazione che mostra il tasso di accuratezza per ogni classe del dataset;
 - Le matrici di confusione in formato ".png" del set di test e di train;
 - Le immagini dei primi 100 campioni del set di test classificati erroneamente.

Come sono stati interpretati i dati in output

Dopo aver simulato l’addestramento di un classificatore su un dizionario di iper-parametri, oltre a ricevere le 10 migliori combinazioni di iper-parametri, si è ottenuto anche il valore

percentuale attorno al quale poi si sarebbe aggirata l'accuratezza di classificazione con l'addestramento effettivo. Questo valore è stato molto utile in quanto ha permesso di avere un'idea preventiva di quali valori aspettarsi una volta addestrato il modello. I risultati ottenuti dall'addestramento effettivo del modello si sono determinati in base ai valori di accuratezza ottenuti. Questi hanno dovuto soddisfare due criteri principali:

1. Maggiore del 70%;
2. Non uguale al 100%.

Nei casi in cui uno dei criteri non è stato soddisfatto il modello è risultato poco performante. Nel caso di una violazione del primo criterio ci si è trovati davanti ad un modello il cui pool di iper-parametri non permetteva all'algoritmo di imparare dalle esperienze di classificazione sui campioni di addestramento, ciò viene tradotto in risultati disastrosi quando l'algoritmo si è trovato a classificare i campioni di test in base all'apprendimento acquisito. Nel caso di una violazione del secondo criterio, invece ci si è trovati davanti un modello che ha smesso di apprendere, ed ha iniziato ad imparare sistematicamente, andando dunque contro i principi cardine del Machine Learning. L'analisi ha previsto anche la considerazione dell'errore quadratico medio per determinare il tasso di errore nella classificazione del modello.

Questo valore è stato molto utile ad individuare i fenomeni di overfitting e underfitting laddove il modello sembrava apparentemente performante. La formula dell'errore quadratico medio è mostrata di seguito:

$$MSE = \frac{1}{N} \sum_{i=1}^N (Y_i - \hat{Y}_i)^2$$

- \hat{Y}_i corrisponde alla predizione del modello;
- MSE corrisponde alla somma dei quadrati residui di cui viene calcolato il valore medio;

In Figura 3.2 si mostra come i valori del tasso di errore nella classificazione sul set di train e di test ha determinato la presenza di underfitting o overfitting:

- Valore di errore alto su entrambe i set -> UNDERFITTING;
- Valore di errore alto sul set di train e basso sul set di test -> UNDERFITTING;
- Valore di errore basso sul set di train e alto sul set di test -> OVERFITTING;
- Valore di errore basso su entrambe i set -> OK.

Data l'elevata quantità di iper-parametri considerate per ogni modello si è definito basso un tasso di errore che non superasse la soglia del valore 3.

		Errore sul set di train	
		Basso	Alto
Errore sul set di test	Basso	OK	UNDERFITTING
	Alto	OVERFITTING	UNDERFITTING

Figura 3.2: Schema riconoscimento underfitting/overfitting

3.3.4 Punti chiave dell’approccio metodologico all’ esperimento

In Figura 3.3 e 3.4 viene mostrato un grafico riassuntivo della metodologia usata per analizzare il comportamento di un classificatore generico sui set di dati MNIST e F-MNIST. Una volta suddivisi i dataset in due set per apprendimento e test, il primo script in Figura 3.3 ha simulato l’addestramento del modello sul set di train del dataset, e successivamente ha testato il suo apprendimento classificando i campioni del set di test. Tutto questo ripetuto tante volte quante sono le combinazioni possibili tra gli iper parametri del dizionario fornito in input.

L’output risultante ha fornito delle linee guida da seguire per effettuare l’apprendimento vero e proprio con il secondo script 3.4. In questo caso il modello è stato effettivamente addestrato e il suo apprendimento testato sempre sugli stessi set di dati, ma questa volta con una specifica combinazione di iper-parametri. I risultati ottenuti da questo script sono più precisi e dettagliati e sono stati registrati se rientravano nei limiti di accettabilità sopra citati, oppure scartati altrimenti. In entrambi i casi il test si è concluso con il cambio di combinazione e la ripetizione delle operazioni fin quando non sono state testate tutte le combinazioni risultate ottimali dall’esecuzione del primo script.

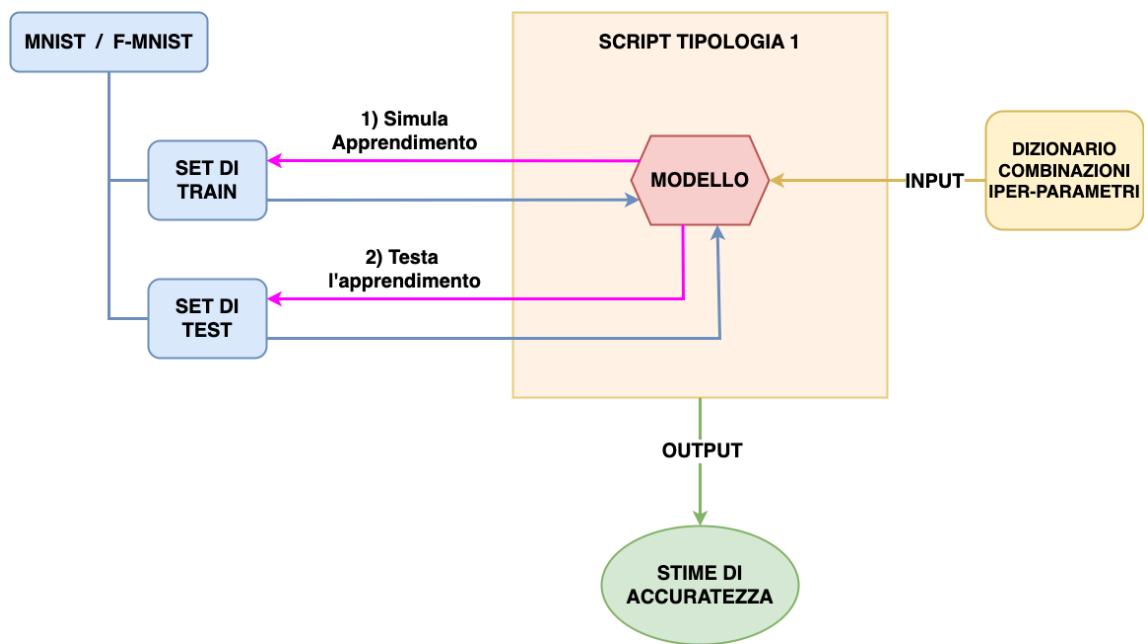


Figura 3.3: Grafico catena metodologica script 1 per analisi delle prestazioni di un classificatore generico

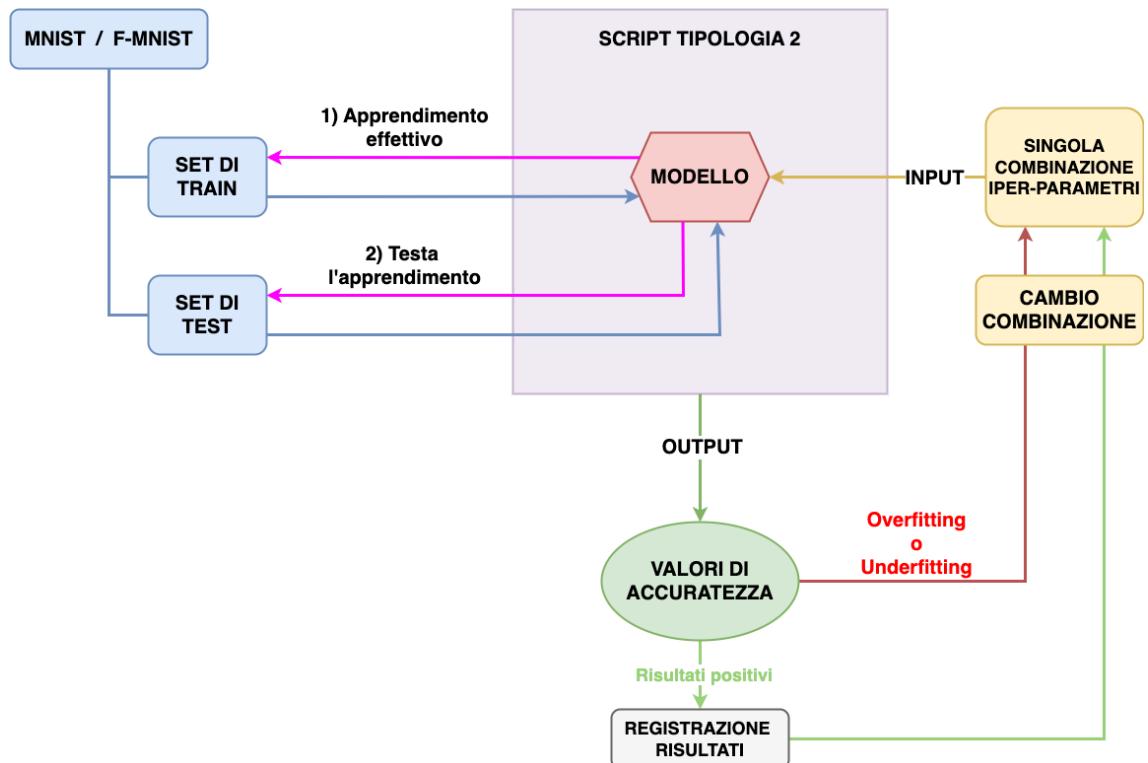


Figura 3.4: Grafico catena metodologica script 2 per analisi delle prestazioni di un classificatore generico

CAPITOLO 4

Risultati

In questo capitolo si esaminano in modo approfondito i risultati ottenuti dalla ricerca al fine di evidenziare come si comportano gli algoritmi selezionati, nel compito di classificare i campioni di immagini forniti da MNIST e F-MNIST, oltre al determinare quale dei due dataset sia realmente il più performante.

4.1 Risultati classificazione con k-Nearest Neighbors

4.1.1 KNN su MNIST

Dall'esecuzione dello script di simulazione è risultato che le migliori 10 combinazioni di iper-parametri in termini di accuratezza prevedono:

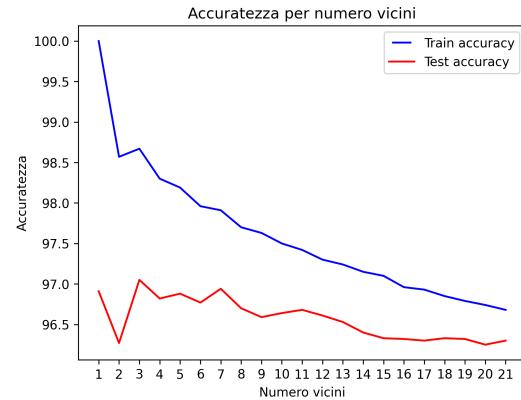
- Un numero di vicini pari a : (3, 5, 7);
- Le metriche di distanza: Euclidea e Minkowski;
- Entrambi i pesi uniforme e distanza.

Inoltre la previsione del tasso di accuratezza sul set di test risultante è pari al 97,05%.

In Figura 4.1a viene mostrato parte dell'output del primo script che evidenzia quanto detto prima sui parametri migliori emersi dalle varie combinazioni. Mentre in Figura 4.1b viene mostrato il grafico di accuratezza per vicini e si può notare di come la linea rossa che rappresenta l'accuratezza sul set di test raggiunga dei picchi in presenza dei punti (3, 5, 7).

```
distance {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}
distance {'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'distance'}
distance {'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}
distance {'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'distance'}
uniform {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}
uniform {'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'uniform'}
uniform {'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}
uniform {'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'uniform'}
distance {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
distance {'metric': 'minkowski', 'n_neighbors': 7, 'weights': 'distance'}
```

(a) Parte dell'output testuale del primo script



(b) Output grafico del primo script

Figura 4.1: Risultati simulazione addestramento KNN su MNIST

Passando infatti all'esecuzione del secondo script si conferma quanto detto prima, infatti su 48 test di addestramento effettivo dell'algoritmo KNN su MNIST i migliori tre risultati si ottengono con i seguenti iper-parametri mostrati nella Tabella 4.1. Da come si può vedere il tasso di accuratezza della migliore combinazione di iper-parametri coincide perfettamente con la previsione generata dallo script di simulazione.

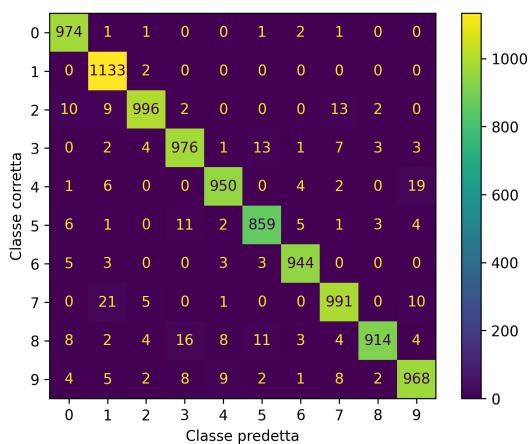
Indice	Parametri	Train Acc	Test Acc
1	{k: 3 , m: minkowsky, w: uniform}	98,67%	97,05%
2	{k: 3 , m: euclidean, w: uniform}	98,67%	96,95%
3	{k: 5 , m: minkowsky, w: uniform}	98,19%	96,88%

Tabella 4.1: Tabella dei migliori tre risultati di addestramento KNN su MNIST

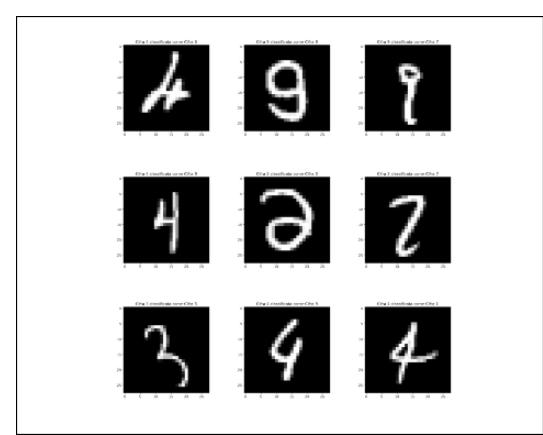
Entrando più nel dettaglio del miglior risultato ottenuto in Figura 4.2a si può osservare la matrice di confusione che mostra i risultati di classificazione per ogni singola classe del dataset. Possiamo notare come la "classe 1" -> "cifra 1" sia quella con la maggiore accuratezza, infatti su 1135 campioni disponibili di test, ben 1133 vengono classificati correttamente,

mentre la "cifra 5" è quella con i risultati peggiori, ma questo è dovuto al fatto è la classe con il minor numero di campioni disponibili 892.

Infine in Figura 4.2b vengono mostrati alcuni dei campioni classificati erroneamente, partendo dalla prima cifra in alto a sinistra abbiamo: cifra 4 classificata come 6, cifra 9 classificata come 8, cifra 9 classificata come 7, cifra 4 classificata come 9, cifra 2 classificata come 0, cifra 2 classificata come 7, cifra 3 classificata come 5, cifra 4 classificata come 9, cifra 4 classificata come 1. Da quello che si può notare la maggior parte degli errori di classificazione sono dovuti all'ambiguità della scrittura delle cifre, alcune che possono ingannare anche l'occhio umano.



(a) Matrice di confusione miglior risultato



(b) Esempi di campioni classificati erroneamente

Figura 4.2: Risultati grafici addestramento effettivo KNN su MNIST

4.1.2 KNN su F-MNIST

I test di simulazione su F-MNIST invece mostrano alcune differenze sostanziali in termini di accuratezza e di iper-parametri. Dall'esecuzione dello script di simulazione è risultato che le 10 migliori combinazioni di iper-parametri in termini di accuratezza prevedono:

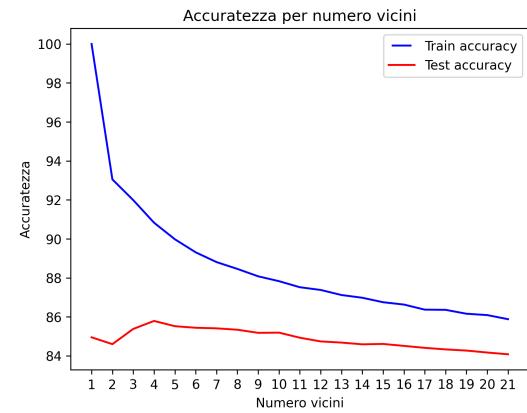
- Un numero di vicini pari a (3, 5, 7, 9, 11, 13);
- Un'unica metrica di distanza: manhattan;
- Entrambi i pesi come per i test su MNIST;

Oltre ad avere un maggior numero di vicini si può notare che il tasso di accuratezza previsto nella classificazione è più basso, ossia dell'86,25%. Da questi primi fattori già si può evincere una maggiore difficoltà di classificazione dei suoi campioni. Nella Figura 4.3b infatti si

può notare come il grafico dell'accuratezza per vicini mostri un andamento più stabile dell'accuratezza sul test, tanto da portare ad aver più vicini che permettono di avere buone prestazioni di classificazione. Mentre in Figura 4.3a viene mostrato parte dell'output grafico relativo alle simulazioni di accuratezza.

```
params
  {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'distance'}
  {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
  {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}
  {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'}
  {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
  {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}
  {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}
  {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'uniform'}
  {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
  {'metric': 'manhattan', 'n_neighbors': 13, 'weights': 'distance'}
```

(a) Parte dell'output testuale del primo script



(b) Output grafico del primo script

Figura 4.3: Risultati simulazione addestramento KNN su F-MNIST

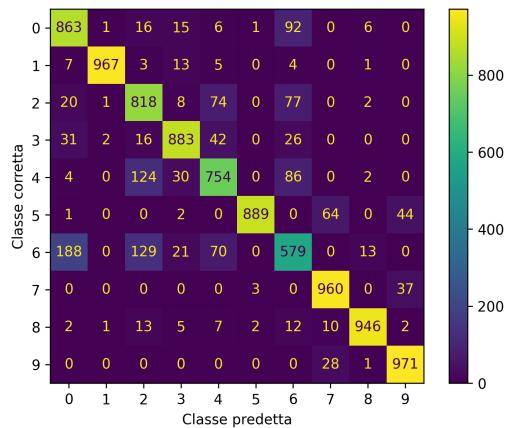
Passando all'esecuzione del secondo script anche in questo caso i risultati preliminari mostrano essere quelli effettivi, che permettono al modello di performare al meglio, infatti in Tabella 4.2 i migliori test ottenuti tra i 48 effettuati mostrano un riscontro con i risultati in Figura 4.3a.

Indice	Parametri	Train Acc	Test Acc
1	{k: 7 , m: manhattan, w: uniform}	88,44%	86,30%
2	{k: 5 , m: manhattan, w: uniform}	90,40%	86,20%
3	{k: 5 , m: manhattan, w: euclidean}	89,98%	85,52%

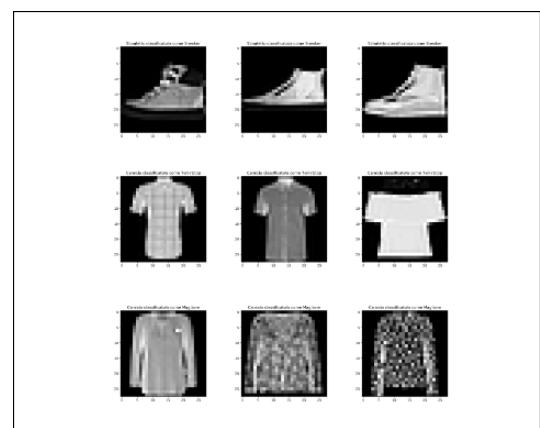
Tabella 4.2: Tabella dei migliori tre risultati di addestramento KNN su F-MNIST

Analizzando anche in questo caso il miglior risultato ottenuto in Figura 4.4a si può vedere la matrice di confusione che mostra come miglior classe riconosciuta quella degli "Stivaletti" con 971 classificazioni corrette su 1000, e i cui unici errori di campionamento si hanno soltanto con la "classe 7" -> "Sneaker", risultato prevedibile data la somiglianza con le immagini mostrate nella prima riga di campioni in Figura 4.4b. In contrapposizione invece abbiamo la "classe 6" -> "Camicia" dove l'algoritmo ottiene i peggiori risultati con soltanto

579 catalogazioni corrette su 1000. I principali errori di classificazione si sono avuti con la "classe 0" -> "T-shirt/top" (seconda riga campioni Figura 4.4b) e la "classe 2" -> "Maglione" (terza riga campioni Figura 4.4b).



(a) Matrice di confusione miglior risultato



(b) Esempi di campioni classificati erroneamente

Figura 4.4: Risultati grafici addestramento effettivo KNN su F-MNIST

4.2 Risultati classificazione con Alberi Decisionali

4.2.1 Alberi Decisionali su MNIST

Dall'esecuzione dello script di simulazione risulta che le 10 migliori combinazioni di iper-parametri in termini di accuratezza prevedono:

- Una profondità dell'albero pari a: (11, 13, 15, 17, 21) livelli;
- Un numero di campioni minimo che può finire nelle foglie finali pari a 1;
- Entrambi i criteri di split "Gini" e "Entropy";
- Entrambe le modalità di scelta del numero di variabili per trovare il best split.

In questo caso abbiamo una varietà più ampia di combinazioni, questo ci mostra come gli Alberi Decisionali siano meno sensibili alle variazioni di iper-parametri, fatta eccezione chiaramente per la profondità dell'albero stesso che come si era preventivato è l' iper-parametro principale. Come si può vedere dal grafico in figura 4.5b il tasso di accuratezza nella classificazione va in crescendo all'aumentare della profondità dell'albero, per poi stabilizzarsi dopo una profondità massima di 10 livelli, e infatti da lì inizia ad ottenere i risultati migliori. Mentre in figura 4.5a viene mostrato parte dell'output testuale del primo script.

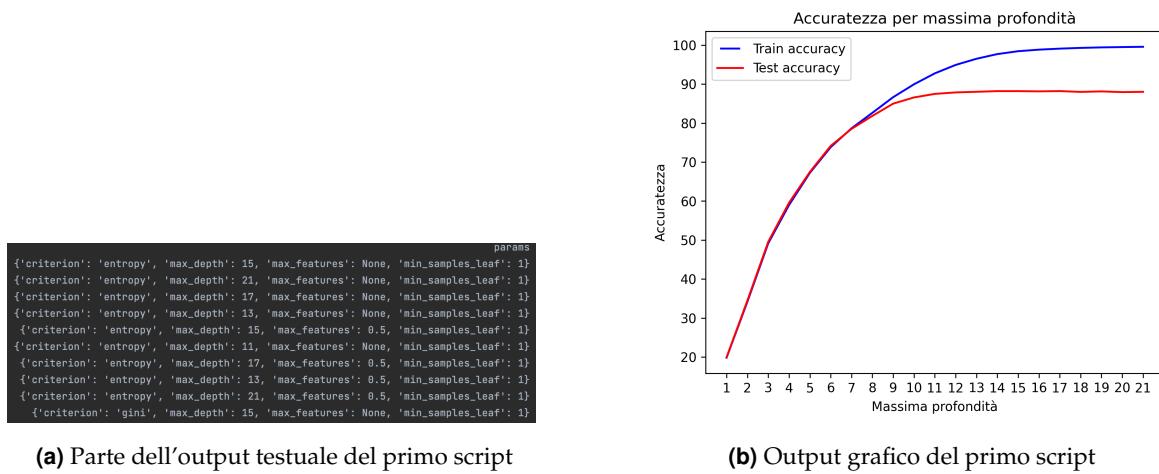


Figura 4.5: Risultati simulazione addestramento Alberi decisionali su MNIST

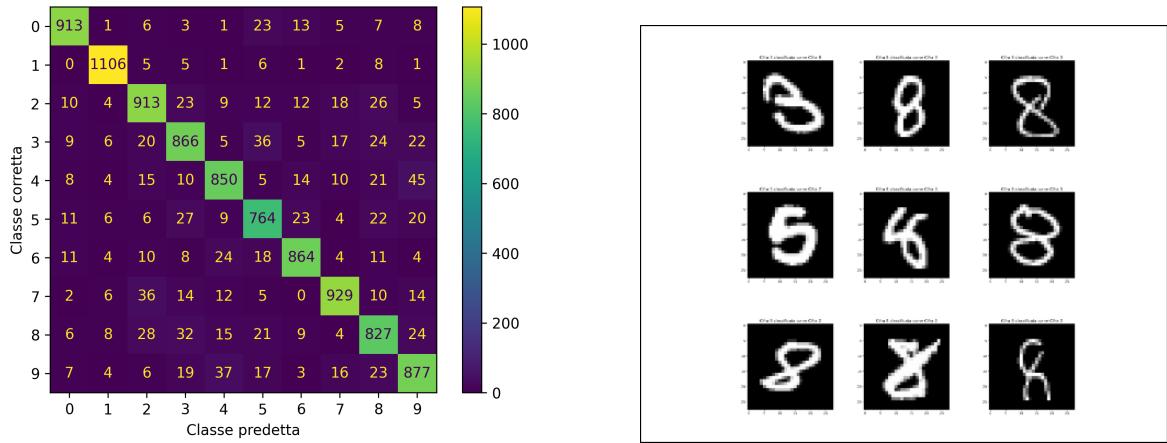
Passando all'esecuzione del secondo script si confermano in parte le stime previste. In Tabella 4.3 vengono mostrati i migliori 3 test sugli 84 effettuati. In particolare nessuno dei tre migliori test ha una configurazione di iper-parametri che risulta tra le prime posizioni delle stime effettuate dal primo script, tuttavia confermano che per ottenere buoni risultati il numero minimo di campioni che può finire nelle foglie deve essere 1.

Indice	Parametri	Train Acc	Test Acc
1	{p: 13 , c: entropy, mf: none, msl: 1}	98,19%	89,09%
2	{p: 17 , c: gini, mf: 0,5, msl: 1}	99,01%	88,51%
3	{p: 15 , c: entropy, mf: 0,5, msl: 1}	99,41%	88,39%

Tabella 4.3: Tabella dei migliori tre risultati di addestramento Alberi Decisionali su MNIST

Analizzando il miglior risultato ottenuto possiamo vedere dalla matrice di confusione mostrata in Figura 4.6a che la classe con la maggior accuratezza di classificazione risulta essere ancora la "classe 1" -> "cifra 1" dove su 1135 campioni disponibili ben 1106 risultano essere classificati correttamente, mentre i peggiori risultati ottenuti si riscontrano con la "classe 8" -> "cifra 8" dove abbiamo solamente 827 campioni classificati correttamente. I principali errori di classificazione si sono avuti con la "classe 3" -> "cifra 3" come mostrato in Figura 4.6b (prima riga di campioni), ma anche con "cifra 5" e "cifra 2" (rispettivamente

seconda e terza riga di campioni). Balzano all'occhio anche i risultati non buoni ottenuti dalla "classe 5", tuttavia è quella con meno campioni disponibili su cui effettuare i test.



(a) Matrice di confusione miglior risultato

(b) Esempi di campioni classificati erroneamente

Figura 4.6: Risultati grafici addestramento effettivo Alberi Decisionali su MNIST

4.2.2 Alberi Decisionali su F-MNIST

Dall'esecuzione dello script di simulazione risulta che le 10 migliori combinazioni di iper-parametri in termini di accuratezza prevedono:

- Una profondità dell'albero pari a: (11, 13, 15, 17, 21) livelli come per i test su MNIST;
- Un numero di campioni minimo che può finire nelle foglie finali pari a 25 (principalmente);
- Entrambi i criteri di split "Gini" e "Entropy";
- Entrambe le modalità di scelta del numero di variabili per trovare il best split.

Da queste prime analisi, i cui risultati testuali sono mostrati in Figura 4.7a, si può notare che la differenza sostanziale è nel numero di campioni minimi che possono finire nelle foglie, (in questo caso 25) dovuto anche alla complessità maggiore del dataset, anche se le uniche 2 simulazioni che prevedono un numero di campioni minimo pari a 1 sono quelle che si piazzano come seconda e quarta. In Figura 4.7b invece si può vedere il grafico di accuratezza sui due set di train e di test che mostra una crescita molto simile a quella avvenuta con MNIST con l'unica differenza di un minor tasso di accuratezza.

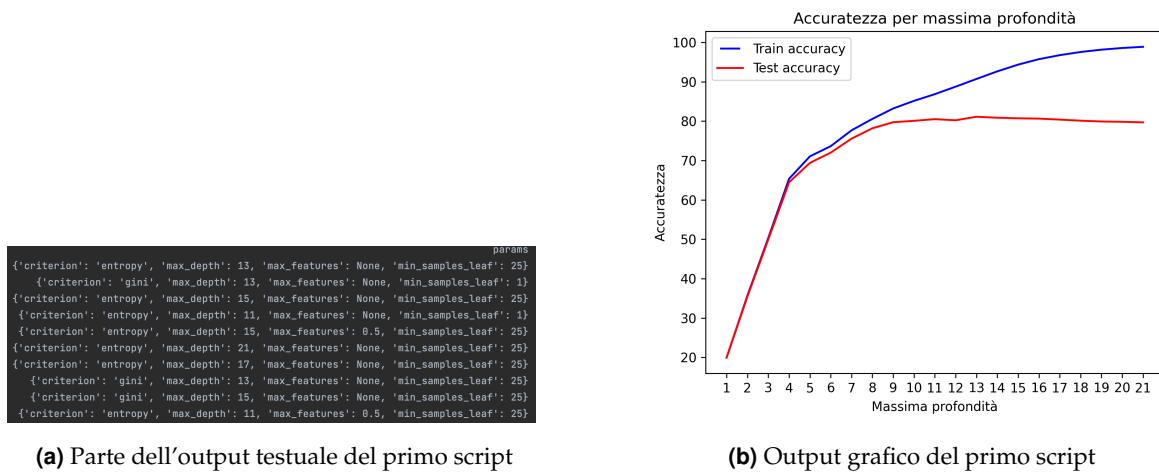


Figura 4.7: Risultati simulazione addestramento Alberi decisionali su F-MNIST

Passando all'esecuzione del secondo script si confermano, in questi casi, le simulazioni effettuate. In Tabella 4.4 vengono mostrati i migliori 3 test sugli 84 effettuati. Si può notare che nonostante dalle simulazioni effettuate ci sia, in modo ricorrente, un numero minimo di campioni che può finire nelle foglie pari a 25, dei tre migliori test di classificazione solo uno riporta questo valore, mentre le migliori classificazioni in termini di accuratezza mostrano ancora un numero di campioni minimi pari a 1. Tuttavia conferma e addirittura supera la previsione di accuratezza sul set di test che dal primo script risulta essere dell'81,45%.

Indice	Parametri	Train Acc	Test Acc
1	{p: 17 , c: entropy, mf: 0,5, msl: 25}	86,76%	81,72%
2	{p: 11 , c: entropy, mf: None, msl: 1}	87,14%	81,53%
3	{p: 11 , c: entropy, mf: 0,5, msl: 1}	87,55%	81,49%

Tabella 4.4: Tabella dei migliori tre risultati di addestramento Alberi Decisionali su F-MNIST

Analizzando i risultati del miglior test in termini di accuratezza, possiamo notare, osservando la Figura 4.9a, di come ci sia più di una classe che classifica in modo ottimo i suoi campioni. Su 10 classi disponibili ben 5 superano la soglia dei 900 campioni classificati in modo corretto, con la migliore che è la "classe 1" -> "Pantaloni" dove su 1000 campioni disponibili ne cataloga correttamente 937. Quella che invece ottiene i peggiori risultati è ancora una volta la "classe 6" -> "Camicia", dove su 1000 campioni ne classifica correttamente

soltanto 558. I principali errori di classificazione si sono avuti con la "classe 0" -> "T-shirt/top" (prima riga campioni Figura 4.9b), la "classe 2" -> "Maglione" (seconda riga campioni Figura 4.9b) e la "classe 4" -> "Cappotto" (terza riga campioni Figura 4.9b).

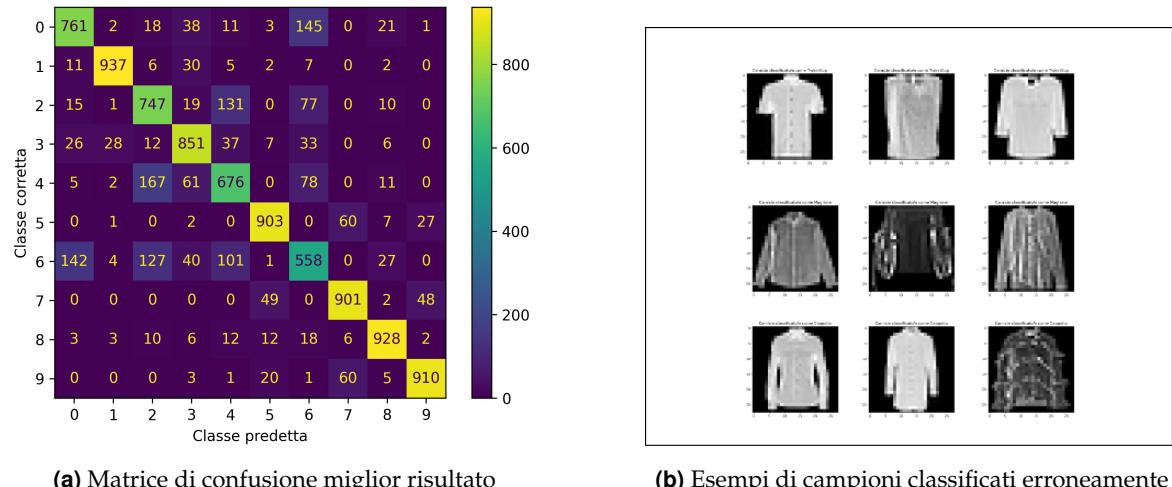


Figura 4.8: Risultati grafici addestramento effettivo Alberi Decisionali su F-MNIST

4.3 Risultati classificazione con Random Forest

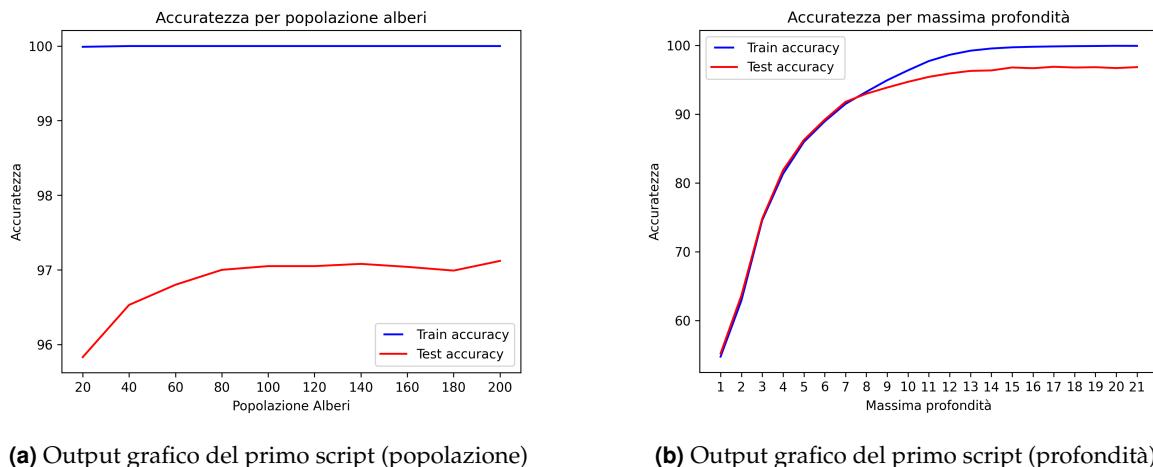
4.3.1 Random Forest su MNIST

Dall'esecuzione dello script di simulazione è risultato che le migliori 10 combinazioni di iper-parametri in termini di accuratezza prevedono:

- Una popolazione di alberi nella foresta pari a : (140, 160, 180, 200);
- Una profondità di ogni albero pari a: 21;
- Un numero di campioni minimo per dividere un nodo intero pari a 2 o 5;
- Entrambi i criteri di split "Gini" e "Entropy".

Da questi risultati preliminari possiamo notare come nelle Random Forest il parametro che più incide sull'accuratezza è il numero di alberi nella foresta, a discapito anche della profondità di un singolo albero, che come possiamo notare è pari soltanto a 21. In Figura 4.9a e 4.9b vengono messi a confronto i due grafici relativi all'accuratezza sul set di train e di test in base al numero di alberi nella foresta (primo grafico) e alla profondità di ogni singolo albero (secondo grafico). Si nota, infatti, che l'accuratezza sul set di train e di test si stabilizzi una volta raggiunta una determinata profondità di ogni singolo albero, la differenza

a quei livelli di accuratezza viene fatta dal numero totale di alberi nella foresta. Come si può vedere nel primo grafico è li che notiamo la vera e propria variazione di accuratezza che oscilla tra il 96% e il 98%. Tale variazione se si osserva il secondo grafico non viene rimarcata, tutt'altro, l'accuratezza di classificazione sembra stabilizzarsi dopo una profondità di ogni singolo albero pari a 11.



"cifra 5" può sembrare quella su cui l'algoritmo ha preformato in malo modo, in realtà mostra buoni risultati considerando che è quella con il minor numero di campioni a disposizione per effettuare i test. La classe peggiore risulta essere quella associata alla "cifra 9" con soltanto 954 campioni classificati correttamente su 1009 disponibili. Non essendoci delle chiare classi di cifre scritte a mano classificate erroneamente, in Figura 4.10b vengono mostrati 9 campioni selezionati e classificati erroneamente a causa della poca chiarezza di questi ultimi.

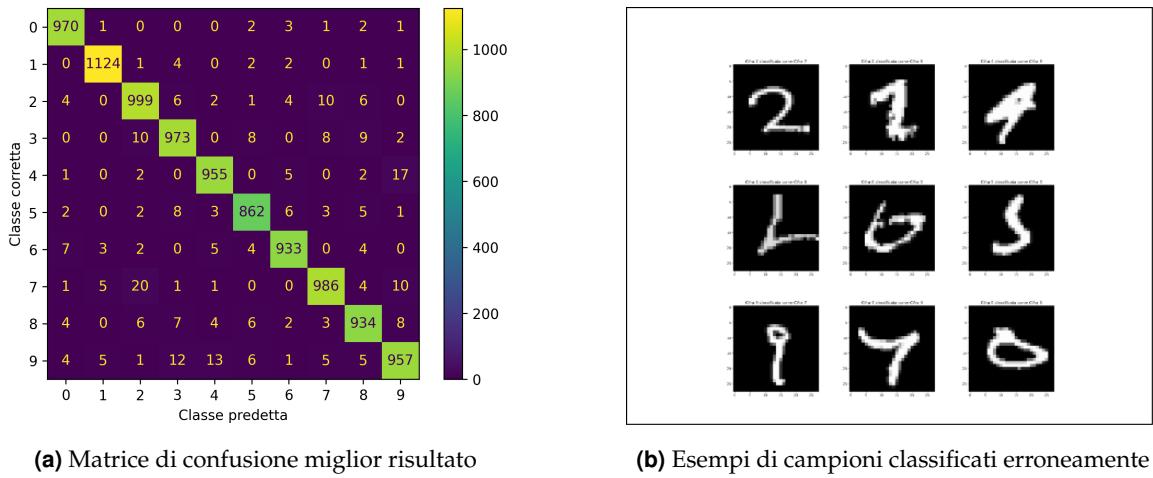


Figura 4.10: Risultati grafici addestramento effettivo Random Forest su MNIST

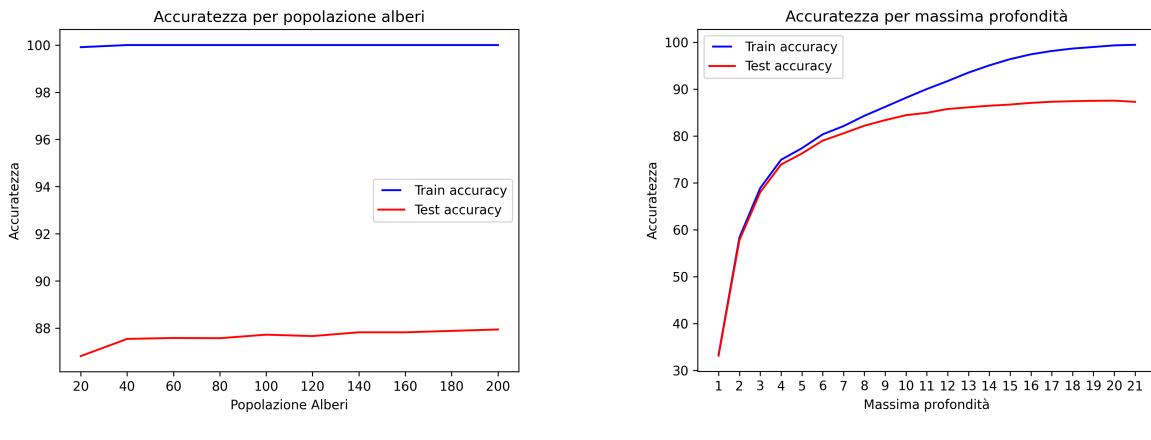
4.3.2 Random Forest su F-MNIST

Dall'esecuzione dello script di simulazione è risultato che le migliori 10 combinazioni di iper-parametri in termini di accuratezza prevedono:

- Una popolazione di alberi nella foresta pari a : (100, 120, 140, 160, 180, 200);
- Una profondità di ogni albero pari a: 21;
- Un numero di campioni minimo per dividere un nodo intero pari a 2 o 5;
- Solo un criterio di split: "Entropy".

Da questi risultati preliminari possiamo notare delle analogie e allo stesso punto differenze con l'analisi effettuata su MNIST. Un'analogia può essere che anche per la classificazione dei campioni di F-MNIST i migliori pool di iper-parametri di simulazione prevedono una profondità per ogni albero pari a 21, una differenza invece può essere nel criterio di spilt per questa classificazione una netta preferenza al criterio Entropy su Gini.

In Figura 4.11a e 4.11b vengono anche qui messi a confronto i due grafici relativi all'accuratezza sul set di train e di test in base al numero di alberi nella foresta (primo grafico) e alla profondità di ogni singolo albero (secondo grafico). Le considerazioni sono analoghe a quelle fatte sulla simulazione su MNIST, solo che in questo caso il tasso di accuratezza è più basso, chiaramente sempre per una questione di maggior complessità dei campioni presenti nel dataset.



(a) Output grafico del primo script (popolazione)

(b) Output grafico del primo script (profondità)

Figura 4.11: Risultati simulazione addestramento Random Forest su F-MNIST

Passando all'esecuzione del secondo script vengono confermate le simulazioni effettuate e discusse precedentemente, tranne per un aspetto che concerne il criterio di split. Come abbiamo visto dalle simulazioni viene preferito come criterio Entropy, ma dai test effettivi uno dei migliori nel suo pool di iper-parametri prevede come criterio Gini. In Tabella 4.6 vengono mostrati i risultati dei 3 migliori test sui 36 effettuati.

Indice	Parametri	Train Acc	Test Acc
1	{ne: 200, p: 21 , c: entropy, mss: 2}	99,86%	87,84%
2	{ne: 200, p: 21 , c: gini, mss: 2}	99,14%	87,62%
3	{ne: 160 p: 21 , c: entropy, mms: 5}	97,64%	87,53%

Tabella 4.6: Tabella dei migliori tre risultati di addestramento Random Forest su F-MNIST

Come si può vedere tutti e tre i test prevedono nel loro pool di iper-parametri una profondità di ogni singolo albero pari a 21. Inoltre il tasso di accuratezza sul set di test per

la prima volta è più basso delle aspettative di simulazione che prevedeva un'accuratezza intorno all'88,44%, che poi dai test è risultata leggermente più bassa di un punto percentuale.

Analizzando i risultati del miglior test in Figura 4.12a si può vedere la matrice di confusione che mostra i migliori risultati sulla "classe 8" -> "Borsa" con ben 973 risultati positivi su 1000 campioni disponibili. In contrapposizione abbiamo ancora una volta la "classe 6" -> "Camicia" dove sui medesimi campioni disponibili ne classifica correttamente soltanto 583. I principali errori si hanno con "classe 0" -> "T-shirt/top" (prima riga di campioni mostrati in Figura 4.12b), "classe 2" -> "Maglione" (seconda riga di campioni mostrati in Figura 4.12b) e "classe 4" -> "Cappotto" (terza riga di campioni mostrati in Figura 4.12b).

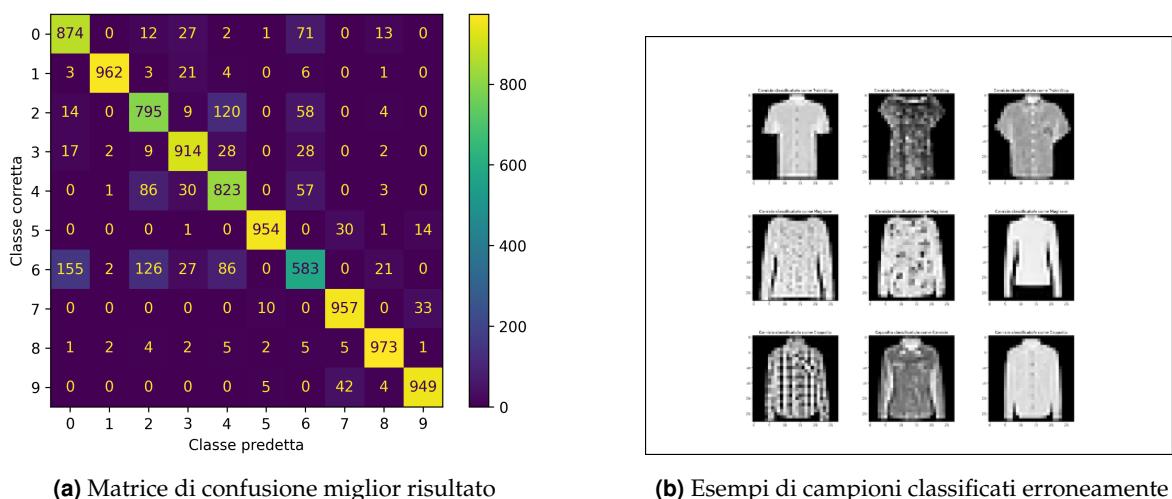


Figura 4.12: Risultati grafici addestramento effettivo Random Forest su F-MNIST

4.4 Risultati simulazione di classificazione con SVM

Per la Macchina a Vettori di supporto i risultati da analizzare sono differenti, dato che su tale algoritmo è stata effettuata solo una procedura di simulazione. Le motivazioni di questa scelta come detto sono dovute all'eccessivo tempo di addestramento di quest'ultimo nel classificare gli innumerevoli campioni di due dataset del calibro di MNIST e F-MNIST. Detto ciò i risultati ottenuti dalla sola simulazione ci permettono di fare delle buone considerazioni sull'accuratezza di classificazione in base al variare dei tre iper-parametri.

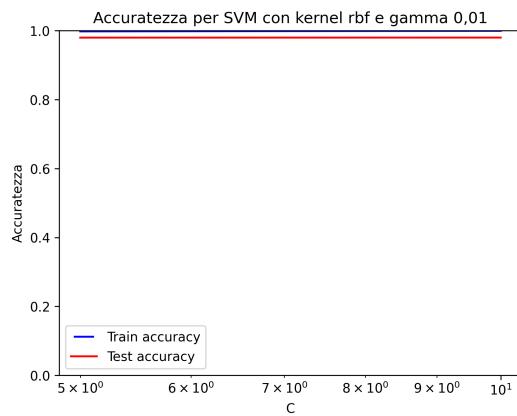
4.4.1 SVM su MNIST

In Tabella 4.7 vengono mostrati i risultati ottenuti in termini di accuratezza sul set di train e di test da parte della SVM al cambio della funzione del kernel.

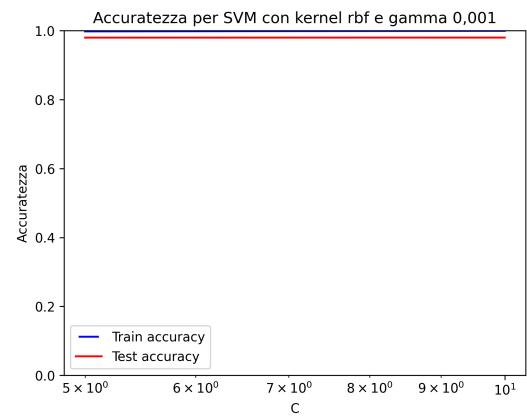
Indice	Funzione del kernel SVM	Train Acc	Test Acc
1	rbf	99,25%	98,00%
2	sigmoid	99,66%	97,64%
3	poly	94,41%	93,65%

Tabella 4.7: Tabella dei migliori risultati per funzione del kernel SVM su MNIST

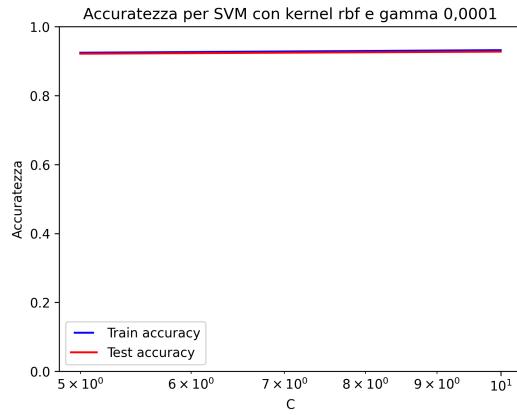
Il risultato dalla miglior simulazione di addestramento dell'algoritmo è quello con funzione kernel "rbf", tuttavia come si può vedere dalle Figure 4.13a e 4.13b il modello impara sistematicamente dato che abbiamo un'accuratezza sul set di train pari al 100%, cosa molto improbabile data anche l'elevata complessità del dataset. Al diminuire di gamma tuttavia notiamo che l'algoritmo migliora le sue prestazioni fino ad ottenere il miglior risultato mostrato graficamente in Figura 4.13c. Inoltre come possiamo notare il tasso di accuratezza rimane pressoché stabile al variare del parametro C che come abbiamo visto rappresenta la massimizzazione del margine di apprendimento della funzione di decisione, questo a prova del fatto che sono principalmente gamma e la funzione del kernel ad influire sostanzialmente sull'accuratezza della classificazione.



(a) Grafico accuratezza set di train e di test con gamma = 0,01



(b) Grafico accuratezza set di train e di test con gamma = 0,001



(c) Grafico accuratezza set di train e di test con gamma = 0,0001

Figura 4.13: Output grafici Simulazione addestramento SVM su MNIST

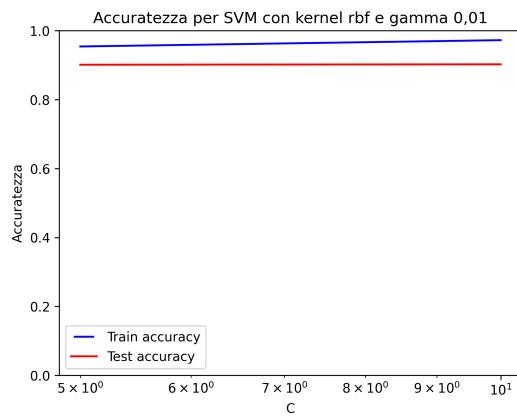
4.4.2 SVM su F-MNIST

Anche in questo caso in Tabella 4.9 vengono mostrati i risultati ottenuti in termini di accuratezza sul set di train e di test da parte della SVM al cambio della funzione del kernel.

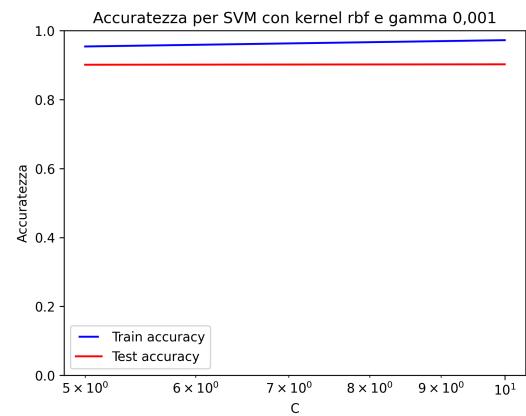
Indice	Funzione del kernel SVM	Train Acc	Test Acc
1	rbf	97,25%	90,25%
2	poly	97,29%	88,22%
3	sigmoid	83,26%	82,98%

Tabella 4.8: Tabella dei migliori risultati per funzione del kernel SVM su F-MNIST

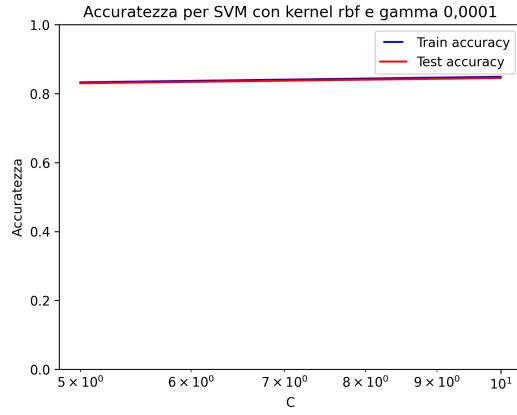
Ancora una volta la funzione kernel è quella che fa performare al meglio il modello anche con F-MNIST. Il risultato dalla miglior simulazione di addestramento dell'algoritmo con funzione kernel "rbf" in questo caso preforma in modo opposto rispetto al suo predecessore MNIST. In questo caso il miglior risultato mostrato nel grafico in Figura 4.14a si ottiene con il valore più alto di gamma, successivamente l'accuratezza di classificazione diminuisce in maniera direttamente proporzionale alla diminuzione del valore di gamma, come mostrato negli altri due grafici rispettivamente nelle Figure 4.14b e 4.14c. Altra differenza sostanziale si può notare nell'importanza che acquista in questo caso il parametro C, che influenza la crescita del tasso di accuratezza, al suo crescere.



(a) Grafico accuratezza set di train e di test con gamma = 0,01



(b) Grafico accuratezza set di train e di test con gamma = 0,001



(c) Grafico accuratezza set di train e di test con gamma = 0,0001

Figura 4.14: Output grafici Simulazione addestramento SVM su F-MNIST

4.5 Risultati classificazione con Percettrone Multinastro

4.5.1 Percettrone Multinastro su MNIST

Dall'esecuzione dello script di simulazione risulta che le 10 migliori combinazioni di iper-parametri in termini di accuratezza prevedono:

- Una ANN con Hidden Layers a 1,2,3 e 4 strati;
- Entrambe le funzioni di attivazione con predominanza di quella Relu su Tanh;
- Entrambe le velocità di apprendimento: costante e adattiva.

Come si può vedere dal grafico generato dalla simulazione, e mostrato in Figura 4.16, l'accuratezza sul set di train e quella sul set di test si muovono in parallelo, e il modello inizia ad apprendere soltanto con una profondità pari a 3 (livelli intermedi). In Figura 4.15 invece viene mostrato parte dell'output testuale.

```
params
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100, 100, 100), 'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100, 100, 100), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'relu', 'hidden_layer_sizes': (100, 100), 'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'relu', 'hidden_layer_sizes': (100, 100), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100), 'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'tanh', 'hidden_layer_sizes': (100, 100), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'relu', 'hidden_layer_sizes': (100, 100, 100, 100), 'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'adam'}
{'activation': 'relu', 'hidden_layer_sizes': (100, 100, 100, 100), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
```

Figura 4.15: Parte dell'output testuale del primo script

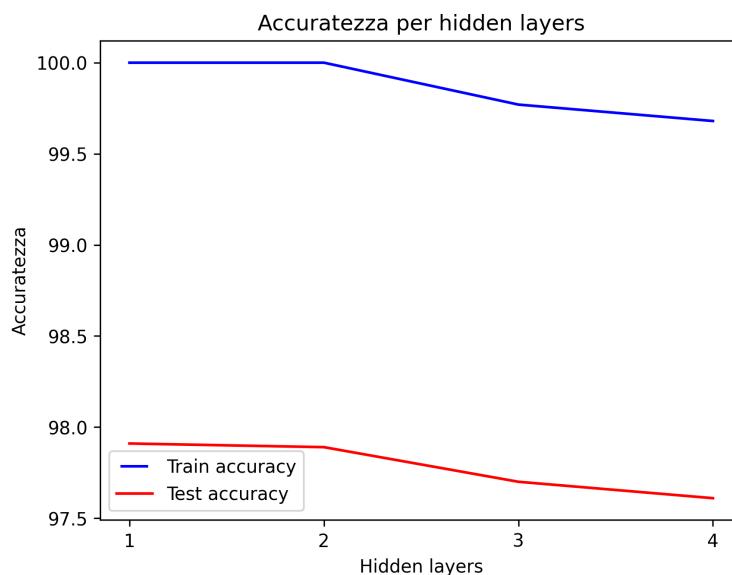


Figura 4.16: Output grafico del primo script (profondità rete)

L'esecuzione del secondo script conferma in parte quanto simulato precedentemente, infatti se osserviamo l'output grafico mostrato in figura 4.15 notiamo che le prime migliori combinazioni di iper-parametri prevedono una funzione di attivazione Tahn. Invece osservando i risultati dell'addestramento effettivo mostrati in Tabella 4.9 evidenziano una netta presenza della funzione di attivazione Relu che permette di ottenere i migliori risultati in termini di accuratezza. Tuttavia in accordo con le simulazioni precedenti il percettrone multinastro performa bene soltanto con una profondità di rete pari a 3 o superiore.

Indice	Parametri	Train Acc	Test Acc
1	{hidden_layers: 3, a: relu, s: adam, lr: adaptive, m: 200}	99,77%	97,70%
2	{hidden_layers: 3, a: relu, s: adam, lr: constant, m: 200}	99,70%	97,64%
3	{hidden_layers: 4, a: relu, s: adam, lr: constant, m: 200}	99,68%	97,61%

Tabella 4.9: Tabella dei migliori tre risultati di addestramento percettrone multinastro su MNIST

Analizzando nello specifico il miglior risultato ottenuto, dalla matrice di confusione mostrata in Figura 4.17a si può notare che la classe che ottiene i migliori risultati è anche in questo caso quella che corrisponde alle "cifra 1" con 1119 campioni classificati correttamente su 1135. Da considerare ottimi anche i risultati ottenuti dalla "classe 2" -> "cifra 2" che su 1032 campioni disponibili ne classifica correttamente 1017, ma anche quelli della "classe 7" -> "cifra 7" che su 1028 campioni ne classifica 1002. Questo è il primo algoritmo che ci fa ottenere degli ottimi risultati su più classi di campioni. La classe che mostra la percentuale di accuratezza più bassa è quella associata alla "cifra 5" ma ciò è dovuto al fatto che è quella che ha meno campioni a disposizione di conseguenza per averne 892 classificarne correttamente 861 è da registrare comunque come un buon risultato. In Figura 4.17b vengono mostrati alcuni campioni classificati erroneamente a causa della loro ambiguità. Partendo dall'alto a sinistra abbiamo: cifra 9 classificata come 5, cifra 2 classificata come 4, cifra 5 classificata come 3, cifra 6 classificata come 0, cifra 8 classificata come 2, cifra 2 classificata come 4, cifra 9 classificata come 4, cifra 7 classificata come 1, cifra 5 classificata come 3.

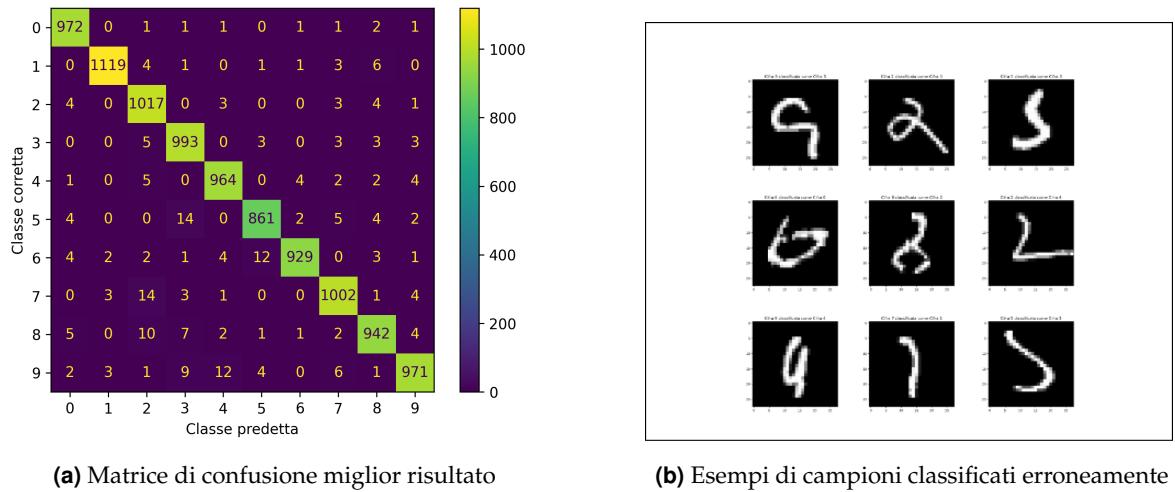
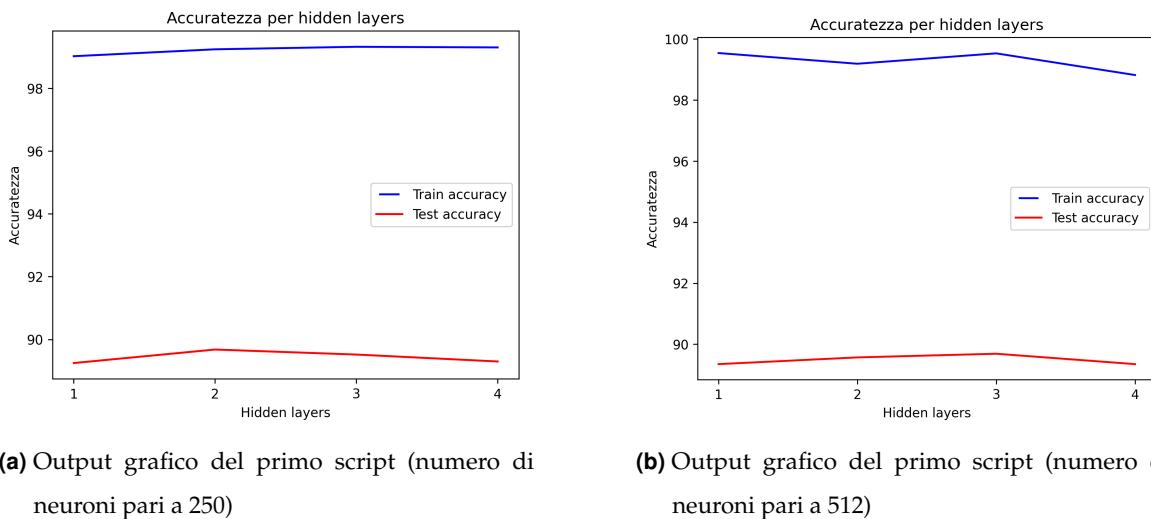


Figura 4.17: Risultati grafici addestramento effettivo Random Forest su MNIST

4.5.2 Percettrone Multinastro su F-MNIST

Il primo script in questo caso viene eseguito due volte per effettuare la simulazione di addestramento della rete su numero crescente di neuroni per ogni livello intermedio al fine di notare se l'aumento dei neuroni, dato la difficoltà del dataset, garantiva un aumento di accuratezza.

I risultati ottenuti, che oltretutto mostrano una combinazione di iper parametri uguale a quella risultata dalle simulazioni con MNIST, confermano quanto affermato. In Figura 4.18a viene mostrato il grafico di accuratezza per la simulazione di addestramento del modello con un numero di neuroni per livello pari a 250. In Figura 4.18b invece il grafico di accuratezza per un numero di neuroni per livello pari a 512. Come possiamo notare l'aumento dei neuroni a prescindere dalla profondità della rete fa lievitare la percentuale di accuratezza del modello anche se di pochi punti percentuali e cambia l'andamento sia del set di train che del set di test. Secondo la simulazione infatti con 250 neuroni per livello avremo un picco di accuratezza con una rete profonda 2 livelli mentre con 512 un picco di accuratezza con una rete profonda 3 livelli. La simulazione nostra un tasso di accuratezza pari all'89,87% con 250 neuroni e dell'89,99% con 512 neuroni.

**Figura 4.18:** Risultati grafici simulazione addestramento percettrone multinastro su F-MNIST

Dall'esecuzione del secondo script vengono confermati quelli che sono i risultati preliminari della simulazione effettuata dal primo script. Guardando la Tabella 4.10, i tre migliori test in termini di accuratezza presentano nelle prime due posizioni una rete con 512 neuroni per livello intermedio, e al terzo posto abbiamo una rete con 2550 neuroni i per livello intermedio. Essa corrisponde alla rete di profondità 2 che come si notava in Figura 4.17a corrispondeva al picco di accuratezza.

Indice	Parametri	Train Acc	Test Acc
1	{hidden_layers: 1, a: tahn, lr: adaptive, neuroni: 512}	99,72%	89,72%
2	{hidden_layers: 3, a: relu, lr: adaptive, neuroni: 512}	99,53%	89,69%
3	{hidden_layers: 2, a: relu, lr: constant, neuroni: 250}	99,24%	89,68%

Tabella 4.10: Tabella dei migliori tre risultati di addestramento percettrone multinastro su F-MNIST

Analizzando il miglior risultato ottenuto dalla matrice di confusione, mostrata in Figura 4.19a si può vedere che le classi che ottengono i migliori risultati sono rispettivamente la "classe 1" -> "Pantaloni" e la "classe 8" -> "Borsa". La prima su 1000 campioni disponibili ne classifica correttamente 979, mentre la seconda sul medesimo pool disponibile ne classifica correttamente 974. In contrapposizione si ha la "classe 6" -> "Camicia" che su 1000 campioni disponibili ne classifica correttamente soltanto 698. I principali errori si hanno anche in questo

caso con le classi "T-shirt/top", "Maglione" e "Cappotto", rispettivamente prima seconda e terza riga di campioni mostrati in Figura 4.19b.

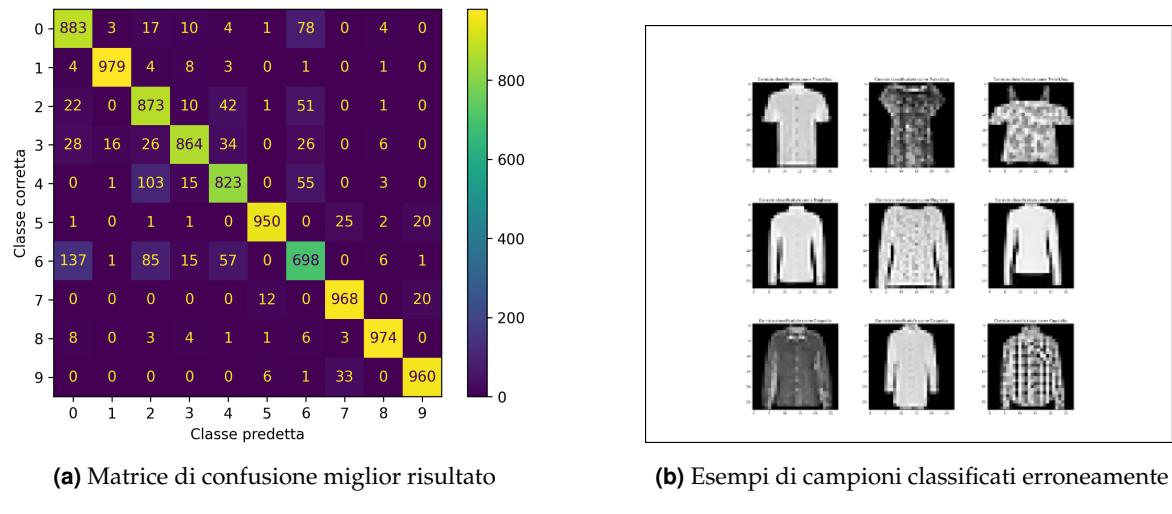


Figura 4.19: Risultati grafici addestramento effettivo perceptron multinastro su F-MNIST

4.6 Confronto finale

In Tabella 4.11 viene mostrato un riepilogo dei test effettuati. Possiamo vedere per i rispettivi dataset i vari algoritmi considerati in ordine di accuratezza di classificazione sul set di test e per ognuno il pool di iper-parametri migliore individuato.

ACCURATEZZA SU MNIST			
Indice	Algoritmo	Train Acc	Test Acc
1	SVM *	99,25%	98,00%
Parametri {fa: rbf, g: 0,0001 }			
2	Percettrone Multinastro	99,77%	97,70%
Parametri {hl: 3, n: 100, fa: relu, s: adam, lr:adaptive, m: 200}			
3	KNN	98,67%	97,05%
Parametri {k: 3, m: minkowski, w: uniform}			
4	Random Forest	99,94%	96,93%
Parametri {ne: 120, p: 21, c: entropy, mss: 2}			
5	Alberi Decisionali	98,19%	89,09%
Parametri {p: 13, c: entropy, mf: none, msl: 1}			

ACURATEZZA SU F-MNIST			
Indice	Algoritmo	Train Acc	Test Acc
1	SVM*	97,25%	90,25%
Parametri {fa: rbf, g: 0,01 }			
2	Percettrone Multinastro	99,72%	89,72%
Parametri {hl: 1, n: 512, fa: tahn, s: adam, lr:adaptive, m: 200}			
3	Random Forest	99,86%	87,84%
Parametri {ne: 200, p: 21, c: entropy, mss: 2}			
4	KNN	88,44%	86,30%
Parametri {k: 7, m: manhattan, w: uniform}			
5	Alberi decisionali	86,76%	81,72%
Parametri {p: 17, c: entropy, mf: 0,5, msl: 25}			
* Effettuato solo un lavoro di simulazione e non un addestramento effettivo sul dataset			

Tabella 4.11: Tabella riassuntiva dei risultati

Come possiamo notare l'accuratezza nella classificazione di campioni è più alta quando gli algoritmi vengono addestrati su MNIST, rispetto a quando l'addestramento avviene su F-MNIST. Questo aspetto evidenzia una maggiore complessità dei campioni presenti nel dataset F-MNIST, il che lo rende più difficile rispetto a MNIST e quindi un degno sostituto di quest'ultimo per addestrare gli algoritmi di intelligenza artificiale.

Altro aspetto da cui si evince la maggior complessità di F-MNIST rispetto a MNIST è quello associato alle classi che ottengono i migliori risultati di classificazione e quella invece che ottiene i peggiori. Nel caso di MNIST la miglior classe riconosciuta è quella associata alla "cifra 1" e la peggiore è quella associata alla "cifra 5". Questo trend si ripete sistematicamente per ogni algoritmo di classificazione che viene addestrato su di esso. Per F-MNIST invece si può notare la presenza di campioni migliori dato che per ogni algoritmo la classe che ottiene il maggior tasso di riconoscimento cambia ogni volta, ciò però non si può dire per la classe associata alle "Camice" che si ripropone come trend negativo per tutti gli algoritmi e che la rende l'unica classe peccante del dataset.

CAPITOLO 5

Conclusioni

5.1 Lavoro svolto

Il lavoro svolto in questa tesi si è incentrato sull’analisi dei più famosi algoritmi di intelligenza artificiale nell’ambito della classificazione quali: KNN, Alberi Decisionali, Random Forest, SVM e Percettrone Multinastro; al fine di individuare i loro principali iper parametri che gli permettono di ottenere i migliori risultati in termini di accuratezza durante il compito di classificazione delle immagini. Il lavoro è cominciato apprendendo le nozioni già presenti in letteratura, da queste si è notato che la maggior parte di essi prevedeva l’analisi di questi algoritmi tenendo in considerazione un solo iper-parametro e vedendo come l’algoritmo si comportava al variare di quest’ultimo. Il lavoro di testi svolto si è prostrato ad effettuare un’analisi più profonda individuando per ognuno degli algoritmi sopra citati un iper-parametro principale e una serie di parametri secondari ma che combinati insieme permettevano di ottenere una combinazione che rendeva l’algoritmo più performante. L’accuratezza di classificazione di questi algoritmi è stata testata su due diversi dataset MNIST e F-MNIST effettuando in parallelo un altro studio ossia quello per determinare quale dei due sia il più difficile e di conseguenza il più adatto ad essere il banco di prova per l’addestramento degli algoritmi di intelligenza artificiale.

Nello specifico sono stati realizzati due script Python per ogni algoritmo:

- Il primo che effettua una simulazione di tutte le possibili combinazioni dato un dizionario di iper-parametri, restituendo le possibili combinazioni ottimali;

- Il secondo invece che effettua il test dell'accuratezza vero e proprio dell'algoritmo su una specifica combinazione di iper-parametri restituendo i tassi di accuratezza rispettivamente per il set di addestramento e per il set di test.

Gli script Python sono stati realizzati facendo uso anche delle librerie scikit-learn che fornivano implementazioni degli algoritmi sopra citati, e di matplotlib per la realizzazione dei grafici visti nei capitoli precedenti utili a leggere, comprendere e interpretare i risultati ottenuti.

5.2 Risultati ottenuti

I risultati ottenuti hanno evidenziato come l'effettiva considerazione di più iper-parametri per ogni singolo algoritmo faccia ottenere degli ottimi tassi di accuratezza. I risultati evidenziano di come tutti gli algoritmi addestrati con una combinazione di iper-parametri raggiungano un'accuratezza di classificazione superiore l'87% per quanto riguarda MNIST, e dell'80% per quanto riguarda F-MNIST.

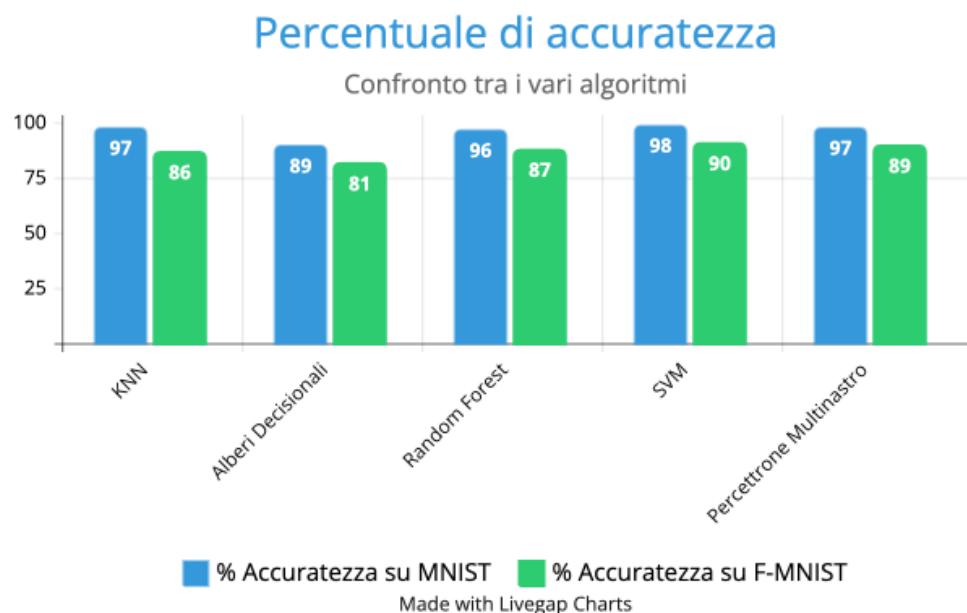


Figura 5.1: Percentuali di accuratezza dei vari algoritmi messe a confronto

Inoltre si è potuto notare come il diretto sostituto di MNIST: F-MNIST sia effettivamente più difficile e quindi un banco di prova più stimolante e accurato per permettere agli algoritmi di AI di essere sempre più efficaci.

5.3 Sviluppi Futuri

Dati i risultati della ricerca si potrebbe procedere a:

- Testare l'apprendimento effettivo dell'algoritmo SVM su MNIST e F-MNIST in modo da ottenere delle stime più accurate,
- Analizzare ancora più a fondo gli iper-parametri di ogni singolo algoritmo al fine di raggiungere la soglia di accuratezza del 90%
- Individuare altri iper-parametri secondari che incidono sull'accuratezza di classificazione degli algoritmi proposti;
- Testare l'accuratezza ci classificazione delle reti neurali convoluzionali sui due dataset MNIST e F-MNIST.

Ringraziamenti

A conclusione di questa ricerca, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe nemmeno.

Innanzitutto ringrazio il mio relatore Fabio Palomba e il mio correlatore Giordano Giammaria, per avermi dato la possibilità di intraprendere questo percorso ed arricchire le mie conoscenze. In questi mesi di lavoro avete saputo guidarmi con consigli e suggerimenti utili, ma soprattutto un ringraziamento va alla vostra infinita disponibilità e puntualità.

Ringrazio mia madre e mio padre per essere stati una guida sincera in tutti i miei momenti di vita. Per aver sostenuto i miei sogni e i miei obiettivi anteponendoli ai loro. Per avermi difeso e amato facendomi sentire realmente la persona più importante della loro vita. Grazie a loro, ad oggi, posso essere orgoglioso della persona che sono diventato.

Ringrazio i miei nonni Balduino, Salvatore, Maria e Ida, per avermi dato con la vostra semplicità: amore e grandi insegnamenti di vita. Per avermi sostenuto in questo percorso interessandovi ad ogni mio risultato. E anche se oggi non siete tutti qui so di avervi resi ugualmente orgogliosi di me.

Ringrazio i miei zii e i miei cugini per rendere speciale la mia famiglia. Non dei semplici parenti ma degli artisti che hanno reso indimenticabili molti momenti della mia vita. Sono felice che mi abbiate trasmesso la passione e l'entusiasmo nel fare ogni cosa, anche le più semplici, e perciò voglio ringraziarvi anche per questo.

Poi voglio ringraziare i miei vicini ormai parte integrante della famiglia: Vincenzo e Stefania, come dei fratelli maggiori, Giovanna e Lello. Un grazie per avermi sempre sostenuto e spronato a raggiungere i miei obiettivi, per questo motivo sono contento di aver condiviso molti momenti belli della mia vita, e spero di continuare a farlo perché, lasciatevelo dire, siete speciali.

Ci tengo a ringraziare la mia ragazza Martina, ho avuto la fortuna e il piacere di averla vicina durante questo percorso, e in tutti gli altri della mia vita quotidiana. Mi hai aiutato e capito nei momenti difficili riuscendo a tirar fuori sempre il meglio di me. Grazie per tutto il tempo che mi hai dedicato e che mi continui a dedicare. Grazie perché ci sei sempre stata e soprattutto grazie per tutta la tranquillità e la gioia che mi dai ogni giorno, anche in quelli in cui non eri al massimo, e grazie per tutti i momenti stupendi che mi fai trascorrere.

Ringrazio Luigi, un fratello che c'è da sempre e continuerà ad esserci per sempre. Una spalla ferma su cui contare, su cui sentirsi forte, con te ho condiviso e so che posso condividere ogni cosa. Per tutte le giornate trascorse, le esperienze fatte insieme e gli aiuti che ci siamo dati durante gli anni e che ci hanno permesso di crescere. Mi è impossibile trovare le parole per esprimere la gratitudine ed il bene che ti voglio, ma lo voglio fare continuando a camminare al tuo fianco.

Non posso non ringraziare Mario per essere stato presente anche in questo ultimo periodo del mio percorso universitario, e per avermi incoraggiato e spronato sempre. Grazie per il tuo affetto sincero e genuino che hai sempre mostrato, da quando ci siamo conosciuti fino a questa parte. Per tutti quei consigli che ci siamo scambiati e la consapevolezza che ci saremo sempre l'uno per l'altro.

Un ringraziamento ad Antonio, Alfonso, Alessandro, Marco che mi hanno accompagnato per gran parte del mio percorso universitario condividendo momenti unici ed indimenticabili. Ma soprattutto i miei ringraziamenti speciali vanno ad Andrea, Carmine, Stefano, Felice, il Presidente, Mario, Gerardo e Mattia. Non si può negare il nostro rapporto speciale, che va oltre l'università e lo studio. Sono contento di vivere con voi tanti bei momenti della mia quotidianità, dallo sfottò calcistico alle grandi uscite. Grazie per essere miei complici, ognuno a suo modo. Sono fortunato che siate capitati tutti voi sul mio cammino.

Ringrazio anche Antonio, Andrea, Massimiliano, Francesco e Luigi per sempre "i ragazzi di Torelli". Grazie per tutti i bei momenti trascorsi insieme, le esperienze nel bene e nel male che abbiamo condiviso e le pagine indelebili che avete scritto nel mio cuore.

Bibliografia

- [1] A. Longo, "Cos'è il Machine learning, come funziona l'apprendimento automatico e quali sono le sue applicazioni," 2022. (Citato alle pagine 4 e 5)
- [2] Ihal, "Come funziona la classificazione delle immagini?," 2021. (Citato a pagina 5)
- [3] C. B. Y. LeCun, C. Cortes, "The mnist database of handwritten digits," 2012. (Citato a pagina 6)
- [4] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017. (Citato alle pagine 7 e 8)
- [5] IBM, "Cos'è l'algoritmo k-nearest neighbors?." (Citato a pagina 14)
- [6] D. Nardini, "Decision Tree algoritmi di machine learning," 2021. (Citato a pagina 17)
- [7] D. Nardini, "Random Forest algoritmi di machine learning," 2021. (Citato a pagina 19)
- [8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. (Citato alle pagine 22 e 26)
- [9] R. Cervelli, "Machine learning,cos'è e come funziona l'apprendimento automatico," 2022. (Citato a pagina 29)
- [10] N. A. Hamid and N. N. A. Sjarif, "Handwritten recognition using svm, knn and neural network," *arXiv preprint arXiv:1702.00723*, 2017. (Citato a pagina 29)

- [11] B. El Kessab, C. Daoui, B. Bouikhalene, M. Fakir, and K. Moro, "Extraction method of handwritten digit recognition tested on the mnist database," *International Journal of Advanced Science & Technology*, vol. 50, pp. 99–110, 2013. (Citato a pagina 31)
- [12] K. Greeshma and K. Sreekumar, "Fashion-mnist classification based on hog feature descriptor using svm," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 5, pp. 960–962, 2019. (Citato a pagina 33)
- [13] D. Vos and S. Verwer, "Efficient training of robust decision trees against adversarial examples," in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 10586–10595, PMLR, 18–24 Jul 2021. (Citato a pagina 35)
- [14] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013. (Citato alle pagine 39 e 40)
- [15] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "K-nearest neighbors classifier documentation," 2022. (Citato a pagina 39)
- [16] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "Decision tree classifier documentation," 2022. (Citato a pagina 39)
- [17] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "Random forest classifier documentation," 2022. (Citato a pagina 40)
- [18] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "Svm classifier documentation," 2022. (Citato a pagina 40)
- [19] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "Mpl classifier documentation," 2022. (Citato a pagina 40)