Università degli Studi di Ferrara

Università degli Studi di Ferrara

Corso di Laurea in Informatica

Applicazione di tecniche di Machine Learning e NLP per l'analisi predittiva del fallimento negli studi clinici

Relatore:

Prof. Marco Alberti

Laureanda:

Chiara Sivieri

Correlatrice:

Prof.ssa Danila Azzolina

Anno Accademico 2023 – 2024

A Luca Sivieri, il mio Papà,
che tu possa essere ricordato in eterno,
tanto quanto saranno eterni
la mia gratitudine ed il mio amore per Te.

Indice

Capitolo 1

Introduzione

Il *Machine Learning* (ML), o apprendimento automatico, rappresenta una macroarea dell'intelligenza artificiale che racchiude una serie di tecniche e approcci sviluppati negli ultimi anni del XX secolo. Questa disciplina è fortemente connessa alla statistica computazionale, la quale si occupa di fare analisi predittive attraverso l'uso di calcoli eseguiti dai computer. È inoltre strettamente collegato a moltissimi campi, come l'ottimizzazione matematica, allo sviluppo di *reti neurali artificiali*, riconoscimento di pattern, all'elaborazione delle immagini, al data mining [1], al filtraggio e algoritmi adattivi. Verranno approfondite nei prossimi capitoli in particolare le tecniche relative alle reti neurali artificiali (capitolo 2.1).

È il 1959 quando Arthur Samuel introduce il termine "Machine Learning" per la prima volta, identificando due approcci distinti. Il primo basatosi su reti neurali, si focalizza nel creare un sistema di apprendimento automatico per impiego generale il cui comportamento è appreso da una rete di commutazione connessa

[1] Il data mining è un insieme di strumenti e algoritmi avanzati che consente ad analisti e utenti finali di risolvere problemi che altrimenti richiederebbero un enorme sforzo manuale o rimarrebbero semplicemente irrisolti[2]

casualmente, seguendo un apprendimento per rinforzo [2]. Il secondo approccio, tuttavia, è largamente più specifico e mirato e riproduce artificialmente una rete neurale ben sviluppata e strutturata, il cui scopo è quello di apprendere solo compiti specifici. Questo metodo risulta essere molto più efficiente del primo dal punto di vista computazionale, richiedendo una supervisione e riprogrammazione continua, dopo ogni nuova applicazione.

Tra i vari obiettivi del machine learning, migliorare le prestazioni degli algoritmi nell'individuazione di pattern all'interno dei dati è certamente il più rilevante per lo scopo di questa tesi. Tuttavia, le tecniche utilizzate per raggiungere tale obiettivo superano i limiti della programmazione tradizionale, poiché permettono alle macchine di apprendere autonomamente dalle informazioni contenute nei dati, senza richiedere istruzioni esplicite.

Gli algoritmi di apprendimento automatico [3], impiegati nelle strategie e tecniche menzionate, stanno ampliando costantemente i loro ambiti di applicazione. In molti casi, sviluppare algoritmi tradizionali per affrontare questi compiti non è solo complesso, ma praticamente impossibile. Alcuni settori di applicazione includono la medicina, il riconoscimento vocale e il filtraggio delle e-mail.

Essendo, come menzionato in precedenza, l' *Apprendimento Automatico* fortemente legato al riconoscimento di pattern e alla teoria computazionale dell'apprendimento, risultano necessari lo studio e la costruzione di algoritmi che possano apprendere da un insieme di dati e fare analisi predittive su di essi, costruendo in modo induttivo un modello basato su dei campioni. Il Natural Language Processing (NLP) risulta essere particolarmente efficace per questo tipo di scopi, da come si può intuire dal nome sono algoritmi di Intelligenza Artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale. Utilizzati

[2] Interagendo direttamente con l'ambiente circostante. Le azioni intraprese vengono valutate attraverso un sistema di "ricompense", un punteggio numerico che incentiva comportamenti desiderabili. Questo approccio, è spesso rappresentato attraverso i processi decisionali di Markov e può essere implementato utilizzando vari tipi di algoritmi. [13].

[3] Ogni algoritmo è progettato per risolvere specifiche tipologie di problemi, come la classificazione, la regressione o il clustering, e la scelta dell'algoritmo dipende dalla struttura dei dati e dagli obiettivi dell'analisi. Tra quelli più degni di nota si identificano i K-NN (k-nearest neighbors), gli alberi decisionali, Random Forest e Gradient Boosting [8]

principalmente per l'analisi di testi dove il processo di vettorizzazione permette di convertire le parole in rappresentazioni numeriche, il che rende i dati per gli algoritmi di ML estremamente più facile e veloci da trattare.

L'obiettivo di questa tesi è quello di cercare di sfruttare al massimo delle loro potenzialità queste ultime caratteristiche del ML: addestrare una macchina in affinché sia in grado di fare un'analisi predittiva sull'esito di esperimenti clinici.

I dati e le informazioni con il quale si è lavorato per raggiungere tale obbiettivo, sono stati presi da un dataset appartenente al database su ClinicalTrials.gov [10], composto da 488.440 righe e 51 colonne, e contenente tutte le informazioni dettagliate sugli esperimenti clinici registrati sul sito. Le informazioni principali includono il titolo dello studio (*Study.Title*), le condizioni mediche (*Conditions*), gli interventi sperimentali (*Interventions*), le misure degli outcome primari e secondari (*Primary* e *Secondary.Outcomes*) e lo stato operativo di ogni studio (Study.Status).

In seguito ad un processo di pulizia dei dati, il dataset è stato ridotto ad 11 colonne fondamentali. Tra queste colonne sono incluse informazioni essenziali come lo stato dello studio, il riassunto (*Brief.Summary*), le condizioni trattate, gli interventi sperimentali, e le caratteristiche demografiche dei partecipanti (*Age, Sex*). Lo stato dello studio è stato convertito successivamente in un valore binario per poter essere utilizzato come *colonna Target* compatibile con il modello. L'*obbiettivo principale*, come precedentemente menzionato, è stato quello di sviluppare un modello predittivo in grado di identificare il fallimento degli esperimenti clinici. Il modello mira ad ottimizzare la gestione degli studi clinici, riducendo il rischio di investire tempo e risorse in esperimenti potenzialmente fallimentari. Per la preparazione dei dati e lo sviluppo del modello sono state utilizzate strategie e tecniche avanzate appartenenti al *NLP*. I risultati ottenuti dall'addestramento e validazione del modello di previsione hanno mostrato un'accuratezza complessiva dell' 83%, con una precisione di circa il 93% per gli studi terminati con successo e una recall del 20% per gli studi falliti. Nonostante l'alta precisione, il basso valore di recall per la classe minoritaria ha indicato che ci sono ampi margini di miglioramento per rilevare in modo più efficace gli studi destinati al fallimento.

Tuttavia, emergono numerose opportunità non solo per il miglioramento, ma anche per l'approfondimento e l'esplorazione delle potenzialità degli studi.

Capitolo 2

Nozioni Preliminari

2.1 Reti Neurali Artificiali

Il neurone è l'unità cellulare fondamentale che costituisce il tessuto nervoso, il quale, insieme alle cellule gliali e al tessuto vascolare, ne contribuisce alla formazione. Grazie alle specifiche proprietà fisiochimiche, è in grado di ricevere, elaborare e trasmettere impulsi nervosi sia eccitatori che inibitori. Nel neurone avvengono diversi processi elettrochimici molto complessi che generano campi elettromagnetici, responsabili dell'elaborazione e trasmissione delle informazioni a livello cellulare [12].

I neuroni sono composti dal soma, ovvero il corpo cellulare, i dendriti, che ricevono segnali da altri neuroni e l'assone, grazie al quale si trasmettono gli impulsi nervosi dal corpo cellulare verso la periferia. Nel cervello umano sono presenti oltre 100 miliardi di neuroni, ciascuno dei quali è interconnesso con circa 10.000 altri neuroni. Queste interconnessioni avvengono attraverso le sinapsi, ovvero i siti di contatto tra le cellule nervose [12].

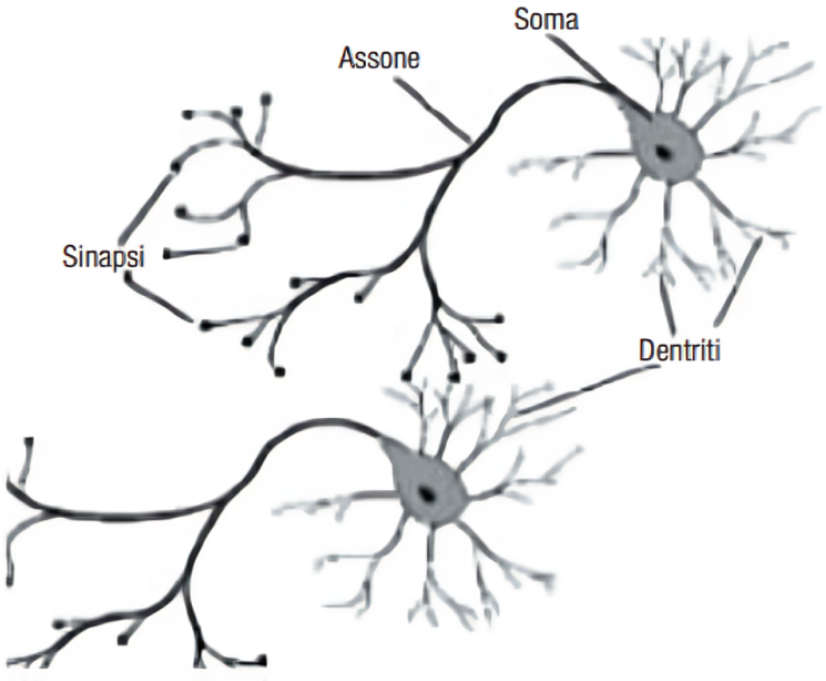

Figura 2.1: Neurone Biologico

I segnali elettrochimici che viaggiano attraverso le sinapsi generano un potenziale elettrico di decine di millivolt e gli impulsi elettrici manifestati si presentano con frequenze intorno ai 100 Hz. Modelli sofisticati descrivono il potenziale di attivazione di un neurone in funzione dei segnali provenienti dalle cellule interconnesse. Questi segnali elettrici sono alla base dell'elaborazione delle informazioni a livello cerebrale.

Lo sviluppo delle reti neurali artificiali è stato fortemente influenzato dagli studi e dalle ricerche sulle reti neurali biologiche presenti nel cervello umano. Attraverso l'analisi del comportamento dei neuroni e delle loro connessioni sinaptiche, i ricercatori hanno gettato le basi per i modelli che vengono utilizzati tuttora. Nello specifico, per la creazione dei modelli matematici sono stati studiati ed analizzati il modo in cui i neuroni biologici ricevono, elaborano e trasmettono segnali elettrici.

Le unità computazionali fondamentali della rete neurale artificiale sono i nodi, interconnessi tra loro, tali connessioni formano un grafo composto da uno strato di

input (che riceve i dati in ingresso, paragonabile all'impulso elettrico per il neurone biologico) e uno strato di output (dal quale escono i dati elaborati). [9]

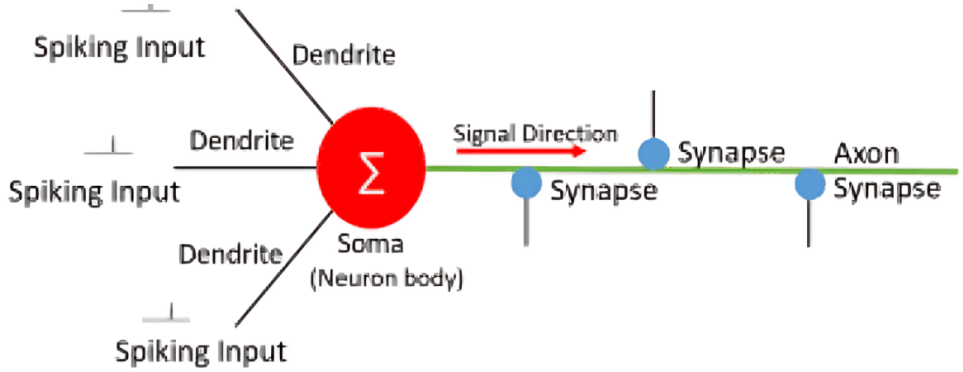

Figura 2.2: Rete Neurale Artificiale a confronto con Neurone.

La trasmissione artificiale, a differenza di quella biologica, avviene mediante numeri che vengono comunicati tramite delle funzioni matematiche. L'elaborazione del dato avviene proprio in questo momento. Sono stati adattati dei metodi di apprendimento automatico di Machine Learning per l'elaborazione dei dati, il cui scopo è quello di addestrare la rete neurale e di migliorare costantemente la sua capacità di elaborazione [9]. Un addestramento efficace porterà ad un algoritmo efficace.

La struttura più semplice, è basata sul modello proposto da McCulloch e Pitts e rielaborato da Rosenblatt, nel quale gli ingressi sono moltiplicati per dei pesi, che rappresentano le connessioni sinaptiche, la somma algebrica dei pesi degli ingressi viene confrontata con un valore di soglia; il neurone fornisce l'uscita 1 se la somma pesata degli ingressi è maggiore del valore di soglia, e l'uscita -1 (o 0) nell'altro caso. [9]

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \equiv g\left(\mathbf{w}^T \mathbf{x} - \theta\right)$$

[1]

È immediato notare che con una scelta di pesi e soglia adeguati, un neurone artificiale (o formali) è in grado di compiere le operazioni logiche elementari

[1] g è la funzione di attivazione del neurone [5]

NOT, AND, OR; questo porta alla possibilità di ricreare le funzioni logiche tramite un'opportuna connessione di neuroni formali. Tuttavia ci sono rappresentazioni più considerevoli di quella menzionata in precedenza.

Il primo esempio significativo di rete neurale artificiale è il percettrone di Frank Rosenblatt nel 1958. Il percettrone è una rete neurale formata essenzialmente da un singolo nodo ricevente input, transitante attraverso una funzione di attivazione processuale e che produce un unico output.

Un'evoluzione di percettrone sono le rete Feedforward: in tali sistemi, l'informazione viaggia in un'unica direzione, cioè dall'input sino all'output. Esistono due tipologie di rete feedforward: rete monostrato, con un unico strato input e output; rete multistrato, con uno o più strati nascosti, ovvero nodi passanti non visibili.

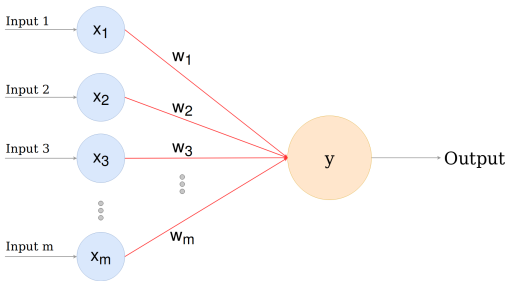

Figura 2.3: Percettrone

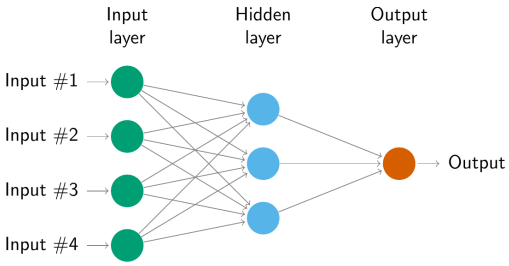

Figura 2.4: Rete feedforward multistrato

In caso di impiego di più livelli nascosti, possiamo parlare di deep learning con reti neurali profonde. Un'altra varietà di rete neurale è data dalle reti neurali ricorrenti (RNN) (capitolo [3.3]). Queste reti neurali sono sostanzialmente simili alle reti multistrato, tranne per il fatto che il loro design permette di rappresentare l'output di alcuni neuroni come input in strati di neuroni precedenti. Ciò aggiunge uno strato di memoria all'algoritmo di apprendimento, che rende le RNN essenzialmente perfette per manipolare sequenze o dati temporali. Una tipologia di reti ricorrenti sono le reti Long Short-Term Memory (LSTM), le componenti principali di una rete neurale LSTM sono lo strato di input sequenziale e lo strato LSTM. Lo strato di input sequenziale consente l'ingresso di dati di sequenza, mentro lo

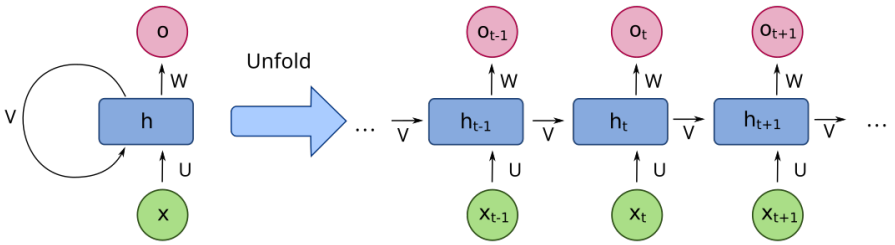

Figura 2.5: Rete Neurale Ricorrente.

strato LSTM è capace di ricordare e collegare informazioni provenienti da diversi intervalli di tempo in una sequenza di dati.

Infine, le reti neurali convoluzionali (CNN), o ConvNet, rappresentano uno degli sviluppi più concreti in termini di IT (Information Technology). Queste sono categoria specifica di rete feedforward multistrato con cinque livelli o più distinti: un livello di input, uno di output e una serie di livelli di strati nascosti. Alcuni strati nascosti si chiamano strati convoluzionali, che eseguono una sorta di elaborazione per generare una mappa di caratteristiche dell'input. Questa mappa è quindi inviata come input allo strato successivo, che consente alla rete di produrre un output maggiormente specificato e dettagliato. A causa del livello di complessità, le CNN sono addestrate e implementate tramite algoritmi di deep learning e possono essere utilizzate per elaborare informazioni complesse, come immagini o video.

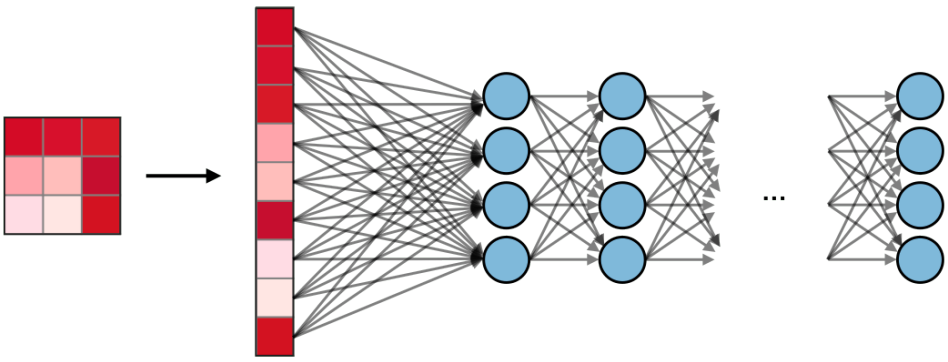

Figura 2.6: Rete Neurale Convunzionale ([1]).

In sintesi, le diverse tipologie di reti neurali si distinguono per il modo in cui sono strutturate e per come vengono applicate. Potendo applicare tecniche di apprendimento avanzate su RNN e CNN, si possono utilizzare in contesti più complessi.

2.2 Il Natural Language Processing

Il Natural Language Processing (NLP) è una branca dell'intelligenza artificiale, che è stata pensata e progettata per permettere ai computer di interpretare, rielaborare, rispondere ed adattarsi al linguaggio umano. Per rendere il tutto possibile sono state unite diverse conoscenze della linguistica computazionale, come ad esempio il riconoscimento vocale e la traduzione automatica [6].

Tra le tecniche principali del NLP si ritrovano:

1. **Tokenization,** utilizzata per il preprocessamento dei dati. Il testo in input viene suddiviso in unità più piccole (ad esempio frasi brevi o singole parole). Questa fase è importante, in quanto rende il tutto più facile da elaborare a livello computazionale;

2. **Parsing e Analisi Sintattica,** rende possibile la comprensione grammaticale del testo in input. In pratica è come se facesse un'analisi logica per permettere al modello di individuare le relazioni soggetto – oggetto – predicato;

3. **Named Entity Recognition (NER),** viene utilizzata per estrarre le informazioni dal testo, tecnica molto utile in casi come quello approfondito in questa tesi;

4. **Stemming e Lemmatization,** il primo riduce le parole alla loro radice e il secondo dopo aver analizzato grammaticalmente la parola, la riduce alla sua forma morfologica di base;

5. **Part-of-Speech (POS) Tagging,** classifica le parole del testo in base alla loro funzione grammaticale. Viene utilizzata per una comprensione migliore del testo;

6. **Sentiment Analysis,** è una tecnica utilizzata spesso per la valutazione automatica delle recensioni, in quanto permette di identificare l'emozione negativa – neutra – positiva espressa in un testo;

7. **Machine Translation (MT),** utilizzata per le traduzioni simultanee.

NLP utilizza modelli architetturali specifici, come le RNN, anche se rischiano di causare problemi con sequenze di linguaggio eccessivamente lunghe. Un'altra opzione valida risulta essere la Long Short-Term Memory (LSTM) (capitolo [3.3]), ovvero una variante delle RNN progettate proprio per gestire il problema del vanishing gradient. È in grado di mantenere informazioni a lungo termine (da qui il nome) all'interno della sequenza. Degni di nota anche i modelli transformer [15], che rendendo possibile al modello di pesare le parole; Generative Pretrained Transformer (GPT) e il Bidirectional Encoder Representations from Transformers (BERT) [3] che hanno migliorato sensibilmente l'universo NLP.

Capitolo 3

Materiali e Metodi

3.1 Descrizione del Dataset

Il dataset originale oggetto delle analisi contiene una raccolta di informazioni relative ad esperimenti clinici registrati su ClinicalTrials.gov [10]. I dati essenziali che esprimono le caratteristiche di ogni esperimento sono contenuti in 51 colonne e 488440 righe: un breve riassunto, le caratteristiche dei soggetti arruolati, gli outcome clinici e, di particolare interesse, lo stato di avanzamento dello studio. La colonna Study.Status difatti, è di notevole rilevanza, e rappresenta il fulcro del dataset, in quanto rende possibile monitorare il progresso ed eventualmente l'esito di ciascuno studio. La colonna assume diversi valori, in base alle caratteristiche e al contesto di ogni esperimento, che comporteranno il successo o meno di ogni trial clinico.

1. **Recruiting**: lo studio è attivamente impegnato nel reclutamento dei partecipanti.
2. **Not yet recruiting**: lo studio non ha ancora avviato il processo di reclutamento.

3. **Enrolling by invitation:** Lo studio seleziona i partecipanti da una popolazione, o gruppo di persone, stabilito in anticipo dai ricercatori.

4. **Completed**: lo studio è stato completato ed i dati necessari sono stati raccolti per l'analisi.

5. **Terminated**: lo studio è stato interrotto prima del previsto, solitamente a causa di problemi logistici, mancanza di risultati o considerazioni etiche.

6. **Withdrawn**: lo studio è stato ritirato prima che fossero avviate le attività di reclutamento o trattamento.

7. **Suspended**: lo studio è stato temporaneamente sospeso, spesso a seguito di questioni di sicurezza o altre circostanze impreviste.

8. **Active, not recruiting**: Lo studio è ancora in corso ma non sta più reclutando nuovi partecipanti.

9. **Unknown**: lo stato dello studio non è chiaro o non è stato aggiornato.

NCT.Number	Study.Title	Study.URL	Acronym	Study.Status	Brief.Summary	Study.Results	Conditions
NCT05013879	al Total Knee Arthroplasty	.gov/study/NCT05013879		COMPLETED	not receiving kinesiotape.	NO	lasty, Replacement, Knee
NCT01402479	in Patients With Migraine	.gov/study/NCT01402479		COMPLETED	ylaxis of migraine attacks.	NO	graine With Hypertension
NCT05600179	ılar Glia After Dex Implant	.gov/study/NCT05600179		COMPLETED	ıasone intravitreal implant	NO	Diabetic Retinopathy
NCT03878979	urrent/Metastatic SCCHN	.gov/study/NCT03878979		COMPLETED	head and neck (SCCHN).	NO	d Neck Cancer Metastatic
NCT04175379	uring One-lung Ventilation	.gov/study/NCT04175379		UNKNOWN	h a constant tidal volume.	NO	Thoracic Surgery
NCT03058679	ission of Crohn's Disease	.gov/study/NCT03058679	DINE-CD	COMPLETED	e this debilitating disease.	YES	Crohn Disease
NCT03877679	al Lichen Planus Patients	.gov/study/NCT03877679		UNKNOWN	eatment (corticosteroids).	NO	Oral Lichen Planus
NCT03960879	) Uterine Cervical Lesions	.gov/study/NCT03960879		UNKNOWN	rmed with the Roche kits.	NO	ous Intraepithelial Lesions
NCT05591079	59 in Subjects With NASH	.gov/study/NCT05591079		COMPLETED	ic Steatohepatitis (NASH)	NO	ic Steatohepatitis (NASH)
NCT01353079	gual-Oral Immunotherapy	.gov/study/NCT01353079		COMPLETED	rt ragweed pollen season.	YES	Allergy

Figura 3.1: Una parte del dataset originale

Oltre alla colonna Study.Status, il dataset completo include ulteriori informazioni genericamente importanti, tra cui il titolo dello studio, un riassunto degli obiettivi, i dati demografici dei partecipanti (in base a sesso ed età), il numero di partecipanti arruolati, le condizioni mediche trattate, gli interventi sperimentali e i metodi utilizzati per la misurazione degli outcome primari e secondari. Alcune delle informazioni nominate rendendono più efficace l'analisi di previsione del risultato degli esperimenti. Sono state infatti selezionate alcune delle colonne

contenenti queste informazioni per mantenerle in un secondo dataset, più pulito, per permettere di avere una visione più chiara degli esperimenti, soprattutto per la macchina.

Listing 3.1: Un estratto del codice Python per una prima pulizia del dataset originale

```
# Selezione delle colonne di interesse e rimozione delle
   righe con Study.Status = 'UNKNOWN'
df2 = df.loc[:, ['Study.Title', 'Study.Status', 'Brief.
   Summary', 'Conditions', 'Interventions', 'Primary.Outcome.
   Measures', 'Secondary.Outcome.Measures', 'Sex', 'Age', '
   Phases', 'Study.Design']]
df2 = df2.drop(df2[df2['Study.Status'] == 'UNKNOWN'].index)
```

Successivamente, il dataset è stato semplificato e ridotto alle informazioni considerate essenziali per un'analisi più mirata. In questa versione pulita, le colonne sono state ridotte a undici campi chiave, mantenendo solo le informazioni centrali per descrivere efficacemente il contenuto di ogni studio clinico. Le colonne selezionate sono state ritenute fondamentali per l'analisi, in quanto forniscono una visione d'insieme degli aspetti più critici di ciascun trial.

Le colonne sono passate a partire da un significativo 54 in totale ad un più gestibile 11. Le colonne selezionate sono le seguenti:

1. **Study.Title**: Il titolo dello studio, che offre una descrizione sintetica dell'argomento e degli obiettivi del trial.
2. **Study.Status**: Lo stato operativo dello studio, che continua a essere la colonna centrale, indicando se lo studio è attualmente attivo, completato o interrotto.
3. **Brief.Summary**: Un breve riassunto dello studio, che descrive gli obiettivi principali e le metodologie adottate.
4. **Conditions**: Le condizioni mediche o patologie studiate, che indicano su quali aree della medicina si concentra lo studio.

5. **Interventions**: Gli interventi sperimentati, che includono trattamenti farmacologici, terapie o procedure.

6. **Primary.Outcome.Measures**: Le misure di outcome primario, che rappresentano i parametri principali utilizzati per valutare l'efficacia del trattamento.

7. **Secondary.Outcome.Measures**: Le misure di outcome secondario, che forniscono ulteriori informazioni sugli effetti del trattamento o intervento.

8. **Sex**: Il sesso dei partecipanti ammessi allo studio, che può essere maschile, femminile o entrambi.

9. **Age**: Le fasce d'età dei partecipanti, che possono variare tra bambini, adulti o anziani.

10. **Phases**: La fase clinica dello studio, particolarmente rilevante per studi su nuovi farmaci, che indica a che stadio si trova la sperimentazione (ad esempio, fase 1, fase 2, fase 3).

11. **Study.Design**: Il disegno metodologico dello studio, che descrive come è stato strutturato il trial, se è randomizzato, cieco o aperto.

Un ulteriore cambiamento nel dataset riguarda la colonna *Study Status*, i cui valori sono stati ridotti a due categorie: **'TRUE'** e **'FALSE'**. In questa nuova classificazione, gli studi classificati come **Terminated** sono stati considerati insuccessi, con valore **True**, mentre tutti gli altri stati sono stati aggregati sotto la categoria di successo, indicata con valore **False**.

Listing 3.2: Estratto del codice Python per poter cambiare il valore della colonna Study.Status

```
# Sostituzione dei valori nella colonna 'Study.Status'
df2['Study.Status'] = np.where(df2['Study.Status'] == '
    TERMINATED', True, False)
```

Questa versione pulita del dataset tiene conto di quanto sopra, consentendo di concentrarsi su ciò che è strettamente necessario per analizzare il progresso, l'efficacia e gli esiti degli studi clinici. Ancora una volta, la colonna del dataset che è rimasta centrale, è stata l' attributo *Study.Status*. Come tale, il file di lavoro emendato consente di monitorare in modo preciso lo stato e lo sviluppo degli studi. Inoltre, un ripensamento delle colonne *Interventions, Primary.Outcome.Measures e Secondary.Outcome.Measures* permette di valutare quali trattamenti sono stati testati e con quali risultati. Le caratteristiche demografiche, come *Sex* e *Age*, consentono non solo di comprendere il target della popolazione studiata, ma permettono di scremare ulteriormente il dataset, dividendo gli studi clinici tra i minori e gli adulti. Rendendo il dataset una risorsa versatile e mirata per l'analisi clinica.

La riduzione e organizzazione dei dati ha permesso di focalizzarsi sugli aspetti critici degll'analisi clinica, come ad esempio il valore degli outcome e le caratteristiche demografiche dei partecipanti, senza perdere di vista gli obiettivi principali degli studi stessi. Il dataset risultante è uno strumento utile non solo per l'analisi attuale, ma anche per potenziali approfondimenti futuri, offrendo una base solida per ulteriori studi e applicazioni nel campo della ricerca clinica.

La struttura del dataset finale può fornire una maggiore efficienza nell'analisi dei dati, favorendo una comprensione dei risultati degli studi clinici migliore e fornendo potenzialemente un modello base replicabile per ulteriori ottimizzazioni dei processi di analisi e gestione di dati sperimentali.

3.2 Pre Processamento e Pulizia dei Dati

Il pre-processamento e la pulizia dei dati sono fasi fondamentali nel flusso di lavoro del machine learning, specialmente quando si lavora con dati non ancora trattati. Questi processi servono a trasformare i dati in un formato interpretabile dal modello, garantendo migliori prestazioni e maggiore accuratezza.

Il pre-processamento è la fase che comprende le tecniche che preparano i dati grezzi ad una forma che può essere utilizzata efficacemente da un modello di machine learning o deep learning.

Le fasi del lavoro per il pre-processamento dei dati sono state classificate come segue:

1. **Tokenizzazione del Testo:** la tokenizzazione consiste nel suddividere un testo in unità più piccole chiamate token, che possono essere parole, frasi, o anche parti di parole [4] [1]. Questo processo facilita la comprensione del testo da parte dei modelli di machine learning e deep learning, poiché li trasforma in sequenze discrete di elementi processabili.

Listing 3.3: Estratto del codice Python in cui viene applicata la tokenizzazione

```
1
2  tokenizer = RegexpTokenizer(r"[a-zA-Z0-9]+")
3
4  [...]
5
6  # Tokenizza e padde il testo pre-processato
7  tokenizer = Tokenizer(num_words=10000)
8  tokenizer.fit_on_texts(df_train['processed_text'])
9
10 # Converte il testo in sequenze di token e applica il
       padding
11 X_train_seq = tokenizer.texts_to_sequences(df_train['
       processed_text'])
12 X_train_padded = pad_sequences(X_train_seq, maxlen=100)
13 X_test_seq = tokenizer.texts_to_sequences(df_test['
       processed_text'])
14 X_test_padded = pad_sequences(X_test_seq, maxlen=100
```

2. **Rimozione della Punteggiatura e dei Numeri e Conversione del Testo in Minuscolo:** il testo non ancora processato contiene spesso punteggiatura e numeri che sono irrilevanti per il modello. La loro rimozione non solo aiuta a concentrarsi sui contenuti più significativi, ma anche a ridurre il rumore nei dati. Inoltre, per rendere il vocabolario meno variabile, tutte le

[1] La Tokenizzazione può essere basata su parole, caratteri o subword in base a quale sia lo scopo prefissato

parole vengono convertite in minuscolo. Questo garantisce che la macchina non abbia problemi di *case sensitivity* [2].

```
# Converte il testo in minuscolo, rimuove la
   punteggiatura e tokenizza
df_train['processed_text'] = df_train['combined_text'].
   apply(lambda x: " ".join(tokenizer.tokenize(x.lower())
   ))
```

La rimozione della punteggiatura è possibile, in questo caso grazie all'utilizzo del tokenizer NLTK (Natural Language Toolkit), in quanto è stato configurato per mantenere solamente le sequenze alfanumeriche. Virgole e punti di domanda, ad esempio, sono simboli e non sono compresi nel pattern della regular expression [11] e, di conseguenza, non verranno mantenuti.

```
 # Inizializza il tokenizer per rimuovere la
    punteggiatura
tokenizer = RegexpTokenizer(r"[a-zA-Z0-9]+")
```

3. **Rimozione delle Stop Words:** le stop words sono parole comuni che non aggiungono significato semantico al modello [7]. Rimuoverle migliora l'efficacia del modello riducendo il rumore dei dati. Essendo la lingua del dataset l'inglese, sono state caricate le stop word in quella lingua. Il testo è stato diviso in parole con x.split(). È stata creata una nuova lista di parole non presenti nell'insieme stop.words ed infine sono state unite le parole filtrate in una singola stringa, facendo in modo così di rimuovere le stop words dal testo.

```
# Rimuove le stop words
stop_words = set(stopwords.words("english"))
df_train['processed_text'] = df_train['processed_text'].
   apply(lambda x: " ".join([word for word in x.split()
   if word not in stop_words]))
```

[2]Sensibilità alle maiuscole, ovvero ogni processo di elaborazione del testo considera come differenti parole uguali ma scritte con lettere in formato diverso - maiuscolo o minuscolo

4. **Gestione dei Valori Mancanti:** nonostante la gestione dei valori mancanti sia una fase genericamente importante nel preprocessing dei dati, per il dataset utilizzato rimuovere le righe contenenti dati mancanti avrebbe ridotto in modo non ignorabile la dimensione del dataset, compromettendo l'accuratezza del modello. Inoltre, essendo di numero elevato,avrebbe potuto introdurre bias [16]. Un'altra soluzione sarebbe potuta essere quella di sostituire i valori mancanti con medie o valori arbitrari, ma data la quantità di dati avrebbe rischiato di distorcere la distribuzione originale di essi e introdurre informazioni non attendibili, rendendo di conseguenza il modello inaffidabile.

Si conclude così la fase di pre-processamento.

La pulizia dei dati è essenziale per garantire che i dati siano correttamente formattati e privi di errori che potrebbero influire negativamente sul modello.

Le fasi della pulizia implementate sono state le seguenti:

1. **Rimozione dei Duplicati:** i dati duplicati possono contribuire a causare problemi di overfitting e distorsione delle metriche di performance, è necessario quindi implementare dei controlli e rimuoverli se presenti.

   Listing 3.4: La funzione calcola un hash unico per ciascun documento nel set di addestramento.

```
# Funzione per calcolare l'hash di un vettore
def hash_vector(vector):
    return hashlib.md5(vector.tobytes()).hexdigest()
```

   Gli hash[3] vengono utilizzati per identificare eventuali dati duplicati nel dataset. Ad esempio, se due documenti (vettori) sono identici, avranno lo stesso hash. In questo modo, diventa facile confrontare gli hash per individuare duplicati e gestirli.

[3]Chiamata la funzione di hash, che prende in input dati di qualsiasi dimensione, viene restituito un valore di lunghezza fissa. Questo valore fornisce una rappresentazione sintetica e univoca dei dati.

```
# Calcola gli hash per i documenti nel set di
   addestramento
train_hashes = [hash_vector(vec) for vec in
   X_train_padded]
```

Listing 3.5: Utilizzando gli hash vengono identificati i documenti duplicati.

```
# Controlla i duplicati nel set di addestramento
duplicate_indices = []
unique_hashes = set()
for i, h in enumerate(train_hashes):
    if h in unique_hashes:
        duplicate_indices.append(i)
    else:
        unique_hashes.add(h)
```

Listing 3.6: Nel caso vengano identificati dei duplicati vengono rimossi sia dal set di addestramento sia dalle etichette corrispondenti.

```
# Rimuove i duplicati
if duplicate_indices:
    print(f"Found␣{len(duplicate_indices)}␣duplicates␣in␣
       the␣training␣set.␣Removing␣them...")
    X_train_padded = np.delete(X_train_padded,
       duplicate_indices, axis=0)
    y_train = np.delete(y_train, duplicate_indices, axis
       =0)
```

Ultimo aspetto importante nella pulizia e nel preprocessamento dei dataset è il bilanciamento dei dati, concentrandosi in particolar modo sulle classe significativamente sottorappresentate rispetto ad altre. Uno sbilanciamento elevato può portare il modello a favorire la classe di cui sono presenti più esempi (maggioritarie) e, di conseguenza, a sfavorire la classi minoritarie, portando ad un bias di classe.

I metodi più comuni di bilanciamento sono *Oversampling* e *Undersampling*. Nel primo caso si creano dei dati fittizzi della classe minoritaria fino a bilanciare

i dati. Nel secondo, si rimuovono alcuni esempi della classe maggioritaria fino al raggiungimento dello scopo menzionato in precedenza.

Il dataset sotto studio era estremamente sbilanciato (quasi 20:1) ed è stato quindi necessario applicare diverse tecniche per risolvere tale problema.

Listing 3.7: Vengono uniti due metodi di bilanciamento per cercare di renderli il più efficenti possibili

```
# Applica SMOTE e undersampling per bilanciare ulteriormente il training set
smote = SMOTE(sampling_strategy=0.5, random_state=42)
```

Dopo aver applicato SMOTE[4], la classe minoritaria è aumtentata di $1/2$ del suo valore iniziale. Viene impostato un seed fisso, in modo da garantire che i risultati si possano riprodurre ad ogni esecuzione.

In seguito si applica un'ulteriore tecnica di bilanciamento, l'*Undersampling*, riducendo il numero di campioni della classe maggioritaria. Alla fine di questo processo, il dataset finale sarà bilanciato e uniforme.

```
undersampler = RandomUnderSampler(sampling_strategy='auto', random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train_padded, y_train)
X_train_res, y_train_res = undersampler.fit_resample(X_train_res, y_train_res)
```

3.3 Architettura della Rete Neurale

L'architettura della rete neurale adattata per lo scopo di questa tesi si basa su un modello LSTM (capitolo: [2.1]), una tipologia di rete ricorrente utilizzata principalmente per gestire sequenze di dati, come in questo caso, testi.

Il modello parte ad essere costruito utilizzando l'API Sequential di Keras[5], che consente di creare un modello strato per strato.

[4] metodo di bilanciamento della tipologia Oversampling

[5] L'API Sequential è ideale per la creazione di modelli lineari, nei quali i layers vengono impilati uno dopo l'altro, seguendo un ordine sequenziale.

```
# Funzione per costruire il modello LSTM con Keras Tuner e
   aggiungere piu' strati
def build_model(hp):
    model = Sequential()
```

Successivamente, lo strato *Embedding*, converte i token numerici in rappresentazioni dense, rendendo così possibile la rappresentazione delle relazioni tra le parole, utilizzando uno spazio di vettori in cui ogni singola parola è mappata all'interno di un dato vettore numerico. Pertanto, otteniamo che gli spazi vettoriali correlati a parole con significati e uso in contesti simili vengono mappati vicini insieme, mentre le parole con significati diametralmente opposti vengono mappate molto più distanti. Allo stesso tempo, il modello potrà catturare le relazioni semantiche e sintattiche tra le parole in modo più efficiente. Viene limitato inoltre il vocabolario a 10.000 parole, che verranno convertite in un vettore denso di 128 dimensioni.

```
# Embedding Layer
model.add(Embedding(input_dim=10000, output_dim=128))
```

Qui viene definita la costruzione dinamica del modello, vengono implementati sia strati LTSM che strati densi [6]. Costruiscono una rete flessibile, in cui il numero di strati ed il numero di unità in ogni strato vengono ottimizzati in modo automatico. Grazie all'unione di più strati è possibile gestire le sequenze di dati e renderli più sintetici. Mediante l'uso di *Batch Normalization* e *Dropout* è inoltre possibile ridurre al massimo il rischio di overfitting durante l'addestramento e migliorare la solidità e la velocità di apprendimento del modello.

```
    # LSTM Layers con unita' personalizzabili
    for i in range(hp.Int('num_lstm_layers', 1, 3)):
        model.add(LSTM(units=hp.Int(f'units_lstm_{i}',
           min_value=64, max_value=256, step=32),
           return_sequences=True if i < 2 else False))

    # Dense Layers
    for i in range(hp.Int('num_dense_layers', 1, 3)):
```

[6] Gli strati densi sono elementi fondamentali nelle reti neurali artificiali. Chiamati anche fully connected layers, ogni neurone è collegato a tutti i neuroni dello strato precedente e a tutti i neuroni dello strato successivo.

```
7         model.add(Dense(units=hp.Int(f'units_dense_{i}',
              min_value=64, max_value=256, step=32), activation=
              'relu'))
8         model.add(BatchNormalization())
9         model.add(Dropout(hp.Float('dropout_rate', min_value
              =0.3, max_value=0.6, step=0.1)))  # Aumento del
              dropout
```

Listing 3.8: La funzione *Sigmoid* è particolarmente adatta per le soluzioni che richiedono classificazione binaria (in questo caso Successo ed Insuccesso)

```
1 # Output Layer
2     model.add(Dense(1, activation='sigmoid'))
```

Il *Learning Rate* stabilisce quanto velocemente o lentamente il modello impara. Il tasso di apprendimento viene trattato come un iperparametro; la selezione del valore ottimale viene effettuata durante il processo di tuning degli iperparametri. I valori selezionati per l'intervallo variano da un apprendimento più graduale fino ad uno più accellerato. Il campionamento dei valori viene fatto su scala logaritmica, in modo da consentire una distribuzione più adeguata di essi.

```
1     # Learning Rate
2     learning_rate = hp.Float('learning_rate', min_value=1e-4,
          max_value=1e-2, sampling='log')
3     optimizer = Adam(learning_rate=learning_rate)
4
5     # Compila il modello
6     model.compile(optimizer=optimizer, loss='
          binary_crossentropy', metrics=['accuracy', 'Precision'
          , 'Recall'])
7
8     return model
```

Keras Tuner avvia un processo di ricerca randomizzata per identificare la miglior combinazione di iperparametri[7], e per farlo ottimizza la funzione di perdita sui dati di validazione. Il miglior set di iperparametri verrà selezionato poi dal Tuner.

[7]Variabili di configurazione esterne utilizzate per lo sviluppo dei modelli di ML.

```
# Imposta Keras Tuner per il tuning degli iperparametri
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=10,
    executions_per_trial=2,
    directory='my_dir',
    project_name='tuning_lstm'
)
```

In sintesi, l'implementazione del modello LSTM con un numero di strati e unità personalizzabili ha reso il modello adatto a pacchetti di dati complessi, dando la possibilità di identificare le relazioni tra i dati e di prevenire overfitting. È stato possibile evitare il sovra-allenamento grazie alle tecniche di regolarizzazione quali *Batch Normalization* e i *Dropout*. Inoltre il modello può ottimizzarsi in modo automatico, tramite *Keras Tuner*, migliorando l'efficienza dell'intero processo.

3.4 Tecniche di NLP e Machine Learning

Diverse tecniche di *Machine Learning* e *Natural Language Processing* (NLP) sono state già discusse nei capitoli precedenti di questa tesi. Nella fase di preparazione e pre-processamento del testo (capitolo: [3.2]) sono state implementate tecniche come la tokenizzazione, la rimozione delle stopword e la conversione del testo in minuscolo, rendendo i dati più semplici e adatti ad un modello di rete neurale.

In seguito proprio per lo sviluppo del modello stesso (capitolo: [3.3]) sono state integrate le tecniche di regolarizzazione *Dropout* e *Batch Normalization*, che hanno permesso di prevenire il sovra allenamento di esso.

Nonostante le tecniche approfondite e menzionate finora siano state implementate per la costruzione del modello e la preparazione dei dati da fornirgli, è presente un'ulteriore strategia di Machine Learning con uno scopo diverso e ben preciso: valutare e predirre le prestazioni del modello precedentemente implementato, ovvero la *K-Fold Cross-Validation*.

L'idea che sta alla base della strategia *K-Fold Cross-Validation* è quella di ridurre il rischio di valutazioni falsate dividendo il dataset in n sottodivisioni e

usare ogni parte sia per l'addestramento che per la validazione, in modo ripetuto. L'addestramento avviene con n - 1 sottodivisioni e la validazione sulla divisione rimanente. Il processo avviene n volte, facendo in modo che ogni divisione venga usata almeno n-1 volte per l'addestramento ed una volta per la validazione.

```
# Cross-Validation K-Fold
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

All'interno di un ciclo for, quindi per ogni fold, i dati di training e i dati di test vengono estratti in base agli indici contenuti nelle varibili train.index e val.index. Grazie alla *Cross-Validation* avviene poi la parte cruciale: ad ogni iterazione viene utilizzata una parte diversa del dataset. Permettendo una maggiore sicurezza riguardo le valutazioni non affidabili.

```
X_train_fold, X_val_fold = X_train_res[train_index],
    X_train_res[val_index]
y_train_fold, y_val_fold = y_train_res[train_index],
    y_train_res[val_index]
```

Allo stesso tempo, all'interno di ogni fold, il *Tuner di Keras* (tuner.search()) cerca gli iperparametri migliori, cercando di ottimizzare il modello. Il modello viene addestrato per dieci epoche per ciascun fold, in ogni iterazione di tuning. Il parametro sul quale si basa il tuner per "decidere" su quali siano gli iperparametri migliori è la *Validation Loss* (perdita di valutazione), ovvero la differenza tra le previsioni del modello e le reali etichette sui dati di validazione.

```
# Esegui la ricerca di tuning con Keras Tuner
tuner.search(X_train_fold, y_train_fold, epochs=10,
    validation_data=(X_val_fold, y_val_fold))
```

3.5 Addestramento e Validazione del Modello

L'ultima fase comprende l'addestramento e la valutazione finale del modello. Tutto il lavoro precedentemente svolto è stato fondamentale per ottimizzare al meglio il modello con i migliori iperparametri trovati tramite *Keras Tuner*. A livello generale, questo approccio fa in modo che il modello non solo sia in grado di

imparare dai dati, ma che sia anche in grado di lavorare su dati nuovi, in particolar modo grazie ai controlli per prevenire l'*Overfitting*.

Nella fase di addestramento, il modello viene allenato per 50 epoche sui dati riequilibrati, con una parte dei dati riservata per la validazione interna.

```
history = best_model.fit(X_train_res, y_train_res, epochs=50,
    validation_split=0.2)
```

Una volta completato l'addestramento, il modello viene testato su un set di test indipendente, che non è stato utilizzato durante la fase di addestramento. Durante questo passaggio, si generano previsioni sotto forma di probabilità, e queste vengono convertite in valori binari (0 o 1) basandosi su un valore di soglia di 0.5. Il valore di soglia rappresenta il limite con il quale il modello decide a quale classe assegnare il dato che gli si sta passando in input. Fondamentalmente separa le probabilità della predizione, nel caso siano maggiori del valore di soglia l'istanza viene passata come classe positiva, negativa altrimenti.

```
# Valutazione sul set di test
y_test_pred_proba = best_model.predict(X_test_padded)
y_test_pred = (y_test_pred_proba > 0.5).astype("int32")
```

Tra le varie metriche di valutazione, quella selezionata è l'Accuratezza, ovvero la percentuae di previsioni corrette fatte dal modello sul test set.

```
# Valutazione
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
```

L'ultima fase della valutazione del modello include ulteriori metriche come *Precisione*, *Recall* e *F1-Score*. Rispettviamente rappresentano quante previsioni sono corrette tra tutte quelle risultate positive, quanto il modello è in grado di riconoscre le istanze positive ed infine la media bilanciata tra i due.

Quindi, una volta che l'addestramento sul train set si è concluso, e si sono trovati gli iperparametri migliori per ottimizzare al meglio il modello, si prosegue con la valutazione su un test set indipendente ed è possibile a questo punto valutare le prestazioni del modello.

Capitolo 4

Risultati

4.1 Prestazioni del Modello

Per avere una visione completa della performance e dei risultati del modello, risulta essere prima necessaria una premessa: questa tipologia di modello richiede delle risorse computazionali significative, nonostante non siano comunque state usate le tecniche migliori e più potenti in assoluto. Questo aspetto, non solo ha influito sui risultati, ma rende il progetto anche non adatto a dispositivi con capacità di calcolo e di memoria limitati.

Nonostante l'implentazione di tecniche di puliza e bilanciamento dei dati avanzate, ed una rete neurale sviluppata specificamente per questo caso di studi infatti, il modello mostra ancora alcune difficoltà.

Difficoltà nel riconoscere la classe minoritaria: Un problema costante durante tutte tutte le fasi di test e di sviluppo è stato sicuramente la difficoltà del modello nel riconoscere la classe minoritaria. Nonostante l'utilizzo di teniche avanzate per il bilanciamento dei dati il problema è rimasto evidente, portando a risultati mediocri, e avendo quindi un peso importante e negativo sulle prestazioni del modello.

```
Test Accuracy: 0.8310631371541163
              precision    recall  f1-score   support

           0       0.93      0.88      0.91     10801
           1       0.12      0.20      0.15       872

    accuracy                           0.83     11673
   macro avg       0.53      0.54      0.53     11673
weighted avg       0.87      0.83      0.85     11673
```

Per la classe 0, la maggioritaria, la *Precision* è molto alta, quindi il modello riconosce correttamente la maggior parte dei casi appartenenti a questa classe. Anche il *Recall* è alto, con un valore di 0.88 quindi la macchina è in grado di riconoscere in modo corretto la maggior parte dei casi della classe 0. Il Punteggio di *F-1 score* indica infatti una buona media dei valori.

Tuttavia, per quanto riguarda la classe minoritaria, i risultati non sono altrettanto soddisfacenti.

Si può osservare una *Precision* molto bassa, con un valore di 0.12, il che vuol dire che molte delle predizioni della classe 1 sono risultate poi essere false positive. Il *Recall* ha un valore leggermente migliore, ma comunque basso, un valore di 0.22 indica che il modello riconosce solo il 20% dei reali casi appartenenti alla classe 1. La *Macro Avg* (media macro) [1]trattando entrambe le classi allo stesso modo, ha un punteggio mediocre, molto penalizzato dalla scarsa performance sulla classe minoritaria. Anche se la *Weighted Avg* (media pesata) ha un valore complessivamente migliore (0.87), in realtà esso è dovuto dalla predominanza della classe maggioritaria. Il modello quindi fatica molto a predirre correttamente i dati appartenenti alla classe 1, ciò è genericamente dovuto dallo squilibrio delle classi, che nonostante si sia cercato di minimizzare fa ancora molta influenza.

[1]La Macro avg è la media semplice delle metriche, considera ogni classe come ugualmente importante, anche se una classe ha molti più campioni rispetto all'altra. [14]

Capitolo 5

Conclusioni

Durante lo sviluppo del progetto, la struttura del modello LSTM e le tecniche di Machine Learning implementate hanno ottenuto una rete neurale artificiale funzionante e, a livello teorico, efficiente. Tuttavia, le prestazioni a livello pratico sono risultate mediocri, dopo un'analisi approfondita i fattori che probabilmente li hanno causati sono stati la difficoltà nel riconoscere la classe minoritaria ed i vincoli computazionali. Tale problematicità è probabilemente dovuta inoltre, non solo allo sbilanciamento, ma anche alla complessità dei dati stessi.

Alcuni dei *possibili miglioramenti futuri* possono essere:

- **Implementazione di modelli più avanzati:** una buona ipotesi di avanzamento potrebbe essere l'utilizzo dei modelli Transformer, molto propensi nel generalizzare sulle classi rappresentate di meno;
- **Sviluppare tecniche di data augmentation:** una *parafrasi automatica* o la *traduzione back and forth* potrebbero aumentare la qualità dei dati con i quali si sta lavorando;
- **Metodi di tuning più avanzati**: nonostante *Keras Tuner* abbia dato la possibilità di individuare alcuni degli iperparametri migliori, metodi più

avanzati (ma più impegnativi a livello computazionale) come *Bayesian Optimization* o *Hyperband* potrebbero essere in grado di fornire risultati più precisi.

- **Risorse computazionali migliori:** chiaramente un miglioramento delle risorse hardware, quindi, ad esempio, la possibilità di *utilizzare GPU o TPU*, porterebbe sicuramente alla possibilità di implementare un modello potenzialmente molto più complesso e preciso.

In conclusione, l'analisi e l'applicazione di modelli di Machine Learning in ambito scientifico, e più precisamente clinico, hanno sicuramente *grandissime potenzialità.* La possibilità di sviluppare un modello complesso che sia in grado, in modo affidabile, di predire il fallimento di un esperimento offre l'opportunità di risparmiare moltissimo tempo e risorse, entrambi fondamentali nella ricerca scientifica.

Rendendo le previsioni appena menzionate automatiche, si può portare ad un processo decisionale *rapido* ed *economicamente conveniente*, dando la possibilità di concentrare tempo e sforzi sugli esperimenti con una maggiore probabilità di successo. Sebbene nel corso di questo progetto siano emerse alcune difficoltà, con alcuni dei miglioramenti menzionati in precedenza, questo tipo di applicazione del Machine Learning ha il potenziale per diventare uno strumento molto utile nella ricerca clinica, contribuendo a migliorare in modo significativo l'efficienza e l'efficacia degli esperimenti.

"Poi si rivolse, e parve di coloro
che corrono a Verona il drappo verde
per la campagna; e parve di costoro
quelli che vince, non colui che perde."
- Dante Alighieri

Bibliografia

[1] Shervine Amidi e Afshine Amidi. *Cheatsheet - Convolutional Neural Networks*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks. Accessed: 2024-09-20. 2019.

[2] Adelchi Azzalini e Bruno Scarpa. *Analisi dei dati e data mining*. Springer Science & Business Media, 2009.

[3] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[4] Enrico Giannini. *Operazioni di base per il NLP: preprocessing e tokenizzazione*. https://enricogiannini.com/18/operazioni-di-base-per-il-nlp-preprocessing-e-tokenizzazione/. Accessed: 2024-09-18. 2023.

[5] Luigi Grippo, M Sciandrone et al. "Metodi di ottimizzazione per le reti neurali". In: *Rapporto Tecnico* 8 (2003), pp. 09–03.

[6] Daniel Jurafsky e James H. Martin. *Speech and Language Processing*. 2nd. Pearson, 2008.

[7] Adventures in Machine Learning. *Streamline Text Analytics: Complete Guide to Removing Stop Words in Python*. https://www.adventuresinmachinelearning.com/streamline-text-analytics-complete-guide-to-removing-stop-words-in-python/. Accessed: 2024-09-18. 2023.

[8] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[9] Bertin Martens e Wouter Von der Wielen. "The data economy: why do firms give away their data for free?" In: *Electronic Markets* 32.1 (2021), pp. 187–203. DOI: 10.1007/s12525-021-00475-2. URL: https://link.springer.com/article/10.1007/s12525-021-00475-2.

[10] U.S. National Library of Medicine. *ClinicalTrials.gov*. https://clinicaltrials.gov/. Accessed: 2024-09-18. 2023.

[11] *nltk.tokenize.regexp — NLTK 3.6.5 documentation*. Accessed: 2024-09-18. 2023. URL: https://www.nltk.org/api/nltk.tokenize.regexp.html.

[12] Red. *Neurone*. 2015. URL: https://www.treccani.it/enciclopedia/neurone%5C_(Enciclopedia-Italiana)/ (visitato il 20/09/2024).

[13] Arthur Samuel. *Arthur Samuel (computer scientist)*. Accessed: 2023-09-07. 2023. URL: https://en.wikipedia.org/wiki/Arthur_Samuel_(computer_scientist).

[14] Scikit-learn Developers. *classification_report - scikit-learn documentation*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html (visitato il 20/09/2024).

[15] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[16] Analytics Vidhya. *End-to-End Introduction to Handling Missing Values*. https://www.analyticsvidhya.com/blog/2021/10/end-to-end-introduction-to-handling-missing-values/. Accessed: 2024-09-18. 2021.