Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Magistrale in
Scienze Statistiche

Utilizzo di Large Language Model per la Classificazione Automatica dei Ticket: Caso studio di Pat SRL

Relatore Prof. Emanuele Aliverti
Dipartimento di Scienze Statistiche

Laureando Scquizzato Enrico
Matricola 2039813

Anno Accademico 2023/2024

Indice

Introduzione

Nel mondo del business odierno, l'*IT Service Management* (ITSM), tematica che verrà approfondita nei capitoli successivi, è diventato un fattore di notevole importanza per il successo aziendale che coinvolge tutti i settori. Rispondere alle crescenti aspettative degli utenti di ricevere un servizio IT personalizzato e di alta qualità, infatti, richiede un impegno costante da parte delle aziende che sviluppano prodotti o servizi in questo ambito per rimanere competitive sul mercato.

In questo contesto, soluzioni *data-driven*, integrazioni di sistemi d'intelligenza artificiale (AI), in particolare di *machine learning* (ML), giocano un ruolo fondamentale nel migliorare l'esperienza del cliente e la qualità del servizio. Le tecniche di ML unite a strumenti statistici avanzati consentono alle aziende di analizzare grandi moli di dati, ottenendo preziose informazioni che possono ottimizzare i processi di erogazione dei servizi, migliorare le interazioni con i clienti e offrire esperienze sempre più personalizzate.

L'obiettivo di questo progetto è esplorare il potenziale del ML e della modellazione statistica nel migliorare servizi per l'*IT Service Management* in un contesto reale. In particolare, verranno sviluppati modelli di classificazione automatica di ticket per discriminare *Incident* e *Problem*, entità cardine dell'ITSM.

Lo studio che verrà condotto consisterà in un confronto approfondito tra diversi modelli di classificazione di dati testuali e tra diverse rappresentazioni numeriche dei dati di input con la finalità di trovare, tra le alternative considerate, il binomio input-classificatore che porta alla classificazione migliore, sia in ambito binario (*Incident*), sia in ambito multiclasse (*Problem*).

Il contesto reale a cui si farà riferimento è quello di PAT s.r.l. (Pat SRL (2024)), azienda che da 30 anni opera nello sviluppo di soluzioni software per aziende e istituzioni pubbliche e private.

Capitolo 1

Definizione del problema di ricerca

1.1 Collaborazione con Pat SRL

1.1.1 Panoramica dell'azienda

L'azienda PAT s.r.l., fondata nel 1992, opera nel nel settore *business-to-business* ed eroga ai suoi clienti soluzioni applicative di *IT Service management*, interazione con i clienti e utenti online, *CRM*, *lead management*, automazione dei processi di business e *social collaboration* aziendale. Anche se la sede centrale è situata a Montebelluna (TV), l'azienda ha diversi uffici dislocati in varie città italiane come Lodi, Milano, Roma e Firenze, oltre ad una sede internazionale a Madrid.

L'obiettivo primario di Pat è quello di semplificare la gestione delle informazioni e delle relazioni tra le diverse aree di un'azienda, con conseguente miglioramento dei processi aziendali, in particolare in settori quali il servizio clienti, il *customer care*, il *service desk*, l'interazione diretta e la comunicazione interna. Grazie ai suoi applicativi fortemente personalizzati a seconda delle esigenze specifiche del cliente, PAT ha raggiunto un livello di adattabilità e flessibilità che la rende estremamente competitiva nel mercato. Tale competitività è dimostrata dalla fiducia accordatale da aziende di alto profilo che hanno scelto Pat per soddisfare le loro esigenze informatiche e di gestione aziendale che operano nei più disparati settori come della moda, manifatturiero, farmaceutico, consulenza IT, bancario, assicurativo, casinò online, enti pubblici, etc.

Un momento significativo nella storia dell'azienda è stato giugno 2013, in cui il Gruppo Pat ha stretto una partnership con Zucchetti: un'importante *software house* italiana nota per essere stata pioniera nell'ambito dell'elaborazione delle paghe e che, nel corso del tempo, ha ampliato il proprio raggio d'azione con l'acquisizione di diverse aziende

che forniscono soluzioni diversificate in vari settori in tutto il mondo.

L'*AI Factory - Innovation Lab* è un'iniziativa recente che riunisce ingegneri e sviluppatori di Pat e Zucchetti. Il suo obiettivo primario è quello di sfruttare i più recenti progressi nelle tecnologie AI e ML e integrarli nelle soluzioni Pat, introducendo funzionalità innovative.

1.1.2 Panoramica dei Software PAT

L'architettura *software* dell'azienda è composta da tre servizi, tra loro interconnessi ed integrabili: *HelpdeskAdvanced* (HDA), CXStudio ed ICstudio. Essi, seppur tutti finalizzati all'ottimizzazione dei processi aziendali, sono pensati per aspetti e contesti differenti di *business*. Di seguito ne vengono descritte brevemente caratteristiche e peculiarità.

HelpdeskAdvanced è una soluzione web e *mobile* finalizzata alla strutturazione e alla gestione dell'*IT Service Management*. L'ITSM di Helpdeskadvanced, in qualità di *business driven solution*, guida le aziende nell'allineamento tra IT e business, gestendo l'intero ciclo di vita di questo sistema di pratiche gestionali, consentendo al dipartimento IT di fornire servizi affidabili e soluzioni che vadano incontro alle esigenze dell'azienda, garantendo efficacia ed efficienza nel coordinamento dei processi coinvolti. HDA si basa sulla logica dell'automazione dei processi ed è definito *ITIL compliant* (Axelos (2024)), ovvero che rispetta e promuove un insieme di linee guida per la gestione dei servizi IT, progettato per migliorare l'efficienza, l'efficacia e la qualità dei servizi IT all'interno di un'organizzazione. HDA propone un'esperienza di Service Desk intuitiva e facile da usare, ottimizzando l'esperienza dell'utente, fornendo canali e interfacce che siano facili da navigare e da utilizzare grazie anche alle molteplici integrazioni con software di terze parti possibili. Oltre a gestire tutti i flussi aziendali, segnalazioni, richieste, HDA consente agli utenti di generare report con grafici e statistiche di base permettendo di monitorare l'efficienza e l'efficacia del servizio nel tempo.

CXStudio, accessibile sia in ambiente *Cloud* che locale, serve come framework per l'interazione multicanale interna e con i clienti sfruttando Intelligenza artificiale e *machine learning*. Questo framework incorpora un assistente virtuale, che consente interazioni in tempo reale 24 ore su 24, 7 giorni su 7, su vari canali preferiti dagli utenti come il web, i principali social network, WhatsApp, Microsoft Teams e altro ancora. Con l'utilizzo interno di tecniche di analisi del *Sentiment* e di KPI emozionali CX mira a rispondere nella maniera più efficace alle richieste dei clienti online, cercando di anticiparne i comportamenti ed influenzarne le scelte. Il framework CX Studio consente agli utenti

di creare flussi di dialogo personalizzati *one-to-one* e applicarli senza soluzione di continuità su tutti i canali, guidati dai principi di proattività, ingaggio e coinvolgimento dei clienti. Per mezzo di questo software è inoltre possibile coordinare in modo efficiente le interazioni tra le diverse aree aziendali all'interno di un unico strumento, anche in questo caso, altamente personalizzato.

ICstudio è una piattaforma di Customer Relationship Management (CRM) dedicata alle esigenze delle diverse aree aziendali, dal reparto commerciale al marketing, dal customer care al call center. Questo Software gestionale è pensato per: automatizzare i processi gestiti quotidianamente dal reparto commerciale, supportare le strategie di vendita, attirare e alimentare l'interesse di leads, definendo azioni e task puntuali, proattivi e specifici sulle opportunità. Con IC Studio è possibile anche creare, monitorare e gestire strategie di marketing su più canali, identificarne facilmente i target, creare campagne e azioni mirate e avviare automaticamente nuove strategie in base ai dati osservati nel tempo.

1.2 Descrizione del Problema

Lo studio condotto nei successivi capitoli di questo lavoro è finalizzato al miglioramento di alcune funzionalità dell'applicativo HDA. Ci poniamo quindi nel contesto ITSM e, più nello specifico, nell'ambito dell'*Incident and Problem Management*. La gestione degli *Incident* (IM) e la gestione dei *Problem* (PM) sono due processi chiave nell'ambito della gestione dei servizi IT.

L'IM si occupa della gestione rapida e efficace degli *Incident*. Secondo ITIL: "un *incident* è un'interruzione imprevista di un servizio o il fallimento di un componente di un servizio che non ha ancora influenzato il servizio". Un evento per essere considerato un *incident*, deve causare un fermo nel servizio e deve essere imprevisto. Esempi come il server che si arresta dopo l'orario lavorativo e la manutenzione programmata, quindi, non sono classificati come *incident* in quanto non interrompono direttamente il processo aziendale. Gli *incident* devono essere risolti immediatamente con una correzione permanente, un *workaround* o una correzione temporanea. L'obiettivo principale della gestione degli *incident* è quello di risolvere nel minor tempo possibile il disagio al fine di ripristinare il servizio.

Il PM a sua volta si concentra sull'identificazione del *problem* ovvero, sempre secondo ITIL, "la causa di uno o più *incident*". Il *problem* è inizialmente sconosciuto e risulta da una serie di *incident* che sono correlati aventi origini comuni. La relazione che intercorre

tra le due entità pertanto è la seguente: mentre i *problem* non sono classificati come *incident*, gli *incident* possono sollevare *problem*, soprattutto se si verificano ripetutamente. Per fare riferimento all'esempio precedente, la situazione del server che viene utilizzato solo durante il giorno, l'arresto dopo l'orario d'ufficio è un *problem* perché, anche se non sta attualmente causando una disfunzione nel servizio, potrebbe potenzialmente accadere di nuovo e diventare un *incident*.

Calando sul piano pratico la teoria precedentemente descritta, l'obiettivo specifico di questo studio è quello di migliorare le prestazioni del sistema di HDA che gestisce il ciclo di vita degli *incident* e dei *problem*, ed implementarne nuove funzionalità come l'individuazione automatica di nuovi *problem* aumentando il numero di quelli conosciuti per formare una base informativa che permetta un riconoscimento tempestivo nel caso si ripresentassero.

Il punto di partenza è la creazione attraverso il portale di HDA dei ticket: le unità statistiche dello studio. Il "ticket" è un elemento fondamentale di HDA che rappresenta la registrazione della richiesta. Esso contiene una serie di dati come l'oggetto, il corpo della richiesta, la soluzione, la priorità, lo stato attuale, l'operatore che l'ha preso in carico, la cartella del catalogo dei servizi da cui è stata registrata la richiesta ed altre caratteristiche secondarie. Tra questi, particolare importanza assumono l'oggetto e la richiesta: sono di fatto le variabili testuali che verranno considerate come esplicative in tutto lo studio; a partire da esse verrà definita la matrice dei regressori X. I passaggi successivi alla creazione del ticket possono essere schematizzati in tre fasi strettamente legate tra loro:

1. Il corpo della richiesta viene concatenato all'oggetto creando un'unica variabile testuale la quale, una volta opportunamente ricodificata, verrà data in input ad un modello di classificazione binaria, che determinerà se si tratta di un *incident* o meno. Nel caso non lo fosse si parlerà di ticket di tipo "*Service Request*" e verrà gestito come una richiesta non prioritaria; nel caso contrario, invece, verrà incluso in un database dedicato in cui sono collezionati tutti i ticket di tipo *incident* e verrà gestito cercando una risoluzione tempestiva del disservizio. Questo database è utile per una ri-stima periodica dei parametri del classificatore binario in modo tale che possa adattarsi ai cambiamenti nel tempo che potrebbero subire le tematiche dei ticket.

2. Al sottoinsieme degli *incident* più recenti a cui non è stato associato alcun *problem* noto viene applicato un modello di *Topic Detection* non supervisionato. Esso consiste sostanzialmente in un algoritmo di *clustering* dei testi dei ticket in cui ogni

cluster (topic) identificato si assume possa essere la rappresentazione di uno specifico *problem* sconosciuto. Ogni cluster di ticket candidato viene così ispezionato manualmente da operatori specializzati che decidono se scartarlo o aggiungerlo alla lista di quelli conosciuti, associando ad esso ciascun *incident* del gruppo.
I ticket appartenenti ai cluster scartati invece tornano a far parte dei ticket non associati a *problem*. L'algoritmo di clustering viene applicato periodicamente in modo tale da variare la finestra temporale dei ticket considerati. Questo fatto, congiuntamente al reinserimento dei ticket dei *problem* scartati nei dati di stima, rende anche in questo caso il modello dinamico ed in grado di cogliere l'evoluzione dei dati potendone potenzialmente scovare pattern sconosciuti.

3. In parallelo alla ricerca di nuovi *problem*, a ciascun nuovo ticket classificato in precedenza come *incident* viene applicato un modello di classificazione testuale multiclasse per associare il ticket ad un *problem* noto con un certo grado di fiducia. Se questo grado di fiducia è sufficientemente alto rapportato ad un valore soglia, si sancisce che vi è un *problem* in atto da dover gestire. In caso contrario, il ticket rimane senza alcun *problem* associato e sarà riutilizzato come input periodico dell'algoritmo di clustering.

La struttura di tale sistema può essere riassunta dall'architettura in Figura 1.1. Questo studio si concentrerà sull'implementazione, l'analisi e la valutazione dei modelli di classificazione testuale coinvolti nella prima e nella seconda fase.

1.3 Struttura dei Dataset

Sia per la classificazione binaria che per la classificazione multiclasse vengono considerati dataset riferiti a diverse aziende utilizzatrici del software HDA con l'intento di trovare il modello più indicato al netto del cliente che lo utilizza.

La varibile dipendente `y`, nel contesto della classificazione binaria, è la dicotomica che assume valore 1 se il ticket è un'*incident*, 0 altrimenti ed è frutto di un'operazione di *tagging* manuale da parte di operatori specializzati. Nel caso multiclasse, invece, `y` non rappresenta il vero *problem* associato al ticket in quanto non ne sono ancora presenti di noti. Secondo le indicazioni degli esperti del campo, si considera come variabile categoriale *proxy* l'elemento di catalogo (`CatalogItem`) da cui è scaturita la segnalazione. Si può immaginare infatti il catalogo di HDA come un'alberatura di cartelle e sottocartelle le cui foglie terminali (elementi di catalogo) si riferiscono ad un singolo servizio o gruppo di servizi. Tale scelta è supportata dal fatto che generalmente un *problem* causa *incident*

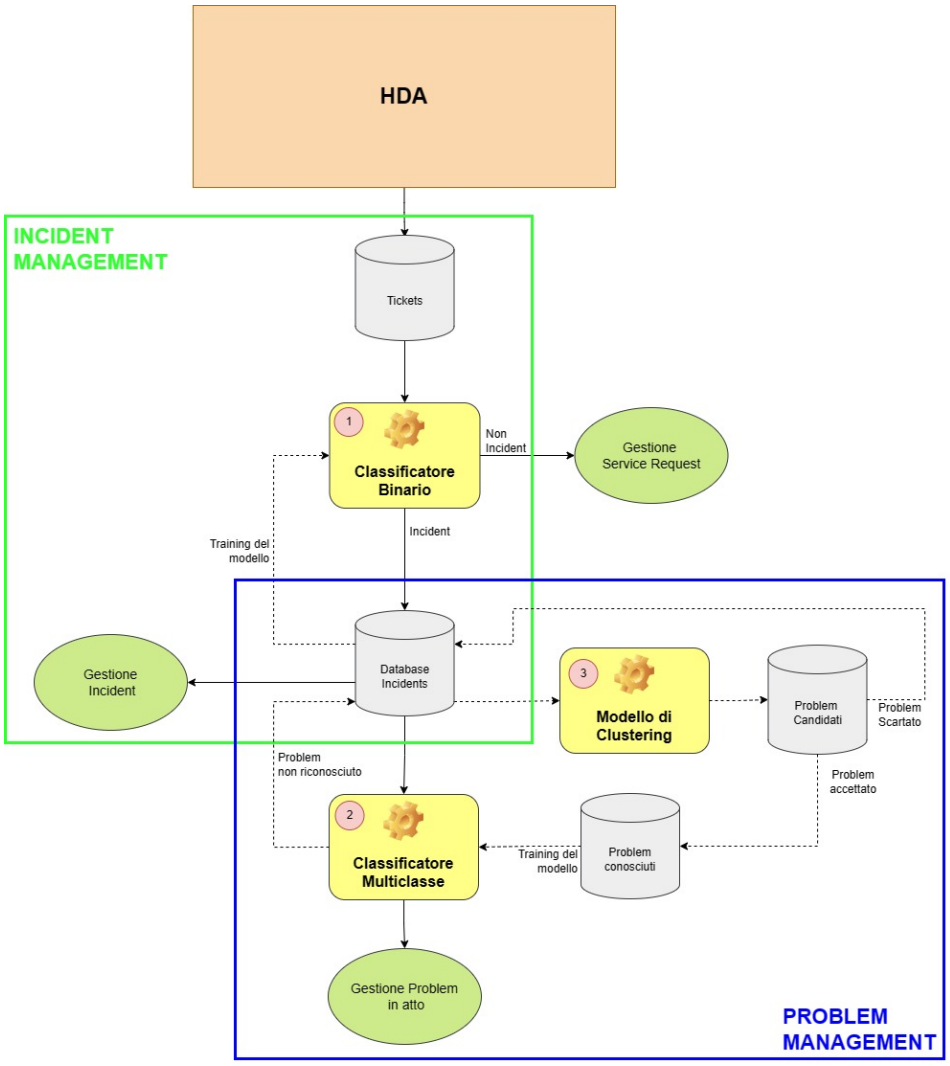

FIGURA 1.1: Architettura generale dell'IM e PM.

che coinvolgono singoli servizi. In questo caso, la cardinalità di `y` varia al variare del dataset.

Ogni osservazione dei dataset si riferisce ad un ticket in lingua italiana per cui sono state registrate, oltre alla variabile dipendente y specifica del tipo di classificazione, numerose covariate tra cui la data di creazione, la data di risoluzione, l'urgenza, l'operatore che l'ha preso in carico, la priorità, l'elemento del catalogo dei servizi da cui proviene ma, in questa analisi, molte non sono d'interesse o si assume non possano contribuire al discriminare gli *incident* o *problem*. Come anticipato in precedenza, infatti, vengono considerate solamente quelle che compongono l'elemento testuale del ticket ovvero `Subject` e `Request` corrispondenti all'oggetto e al corpo della richiesta.

`Subject` e `Request` presentano caratteristiche differenti a livello semantico. Mentre il primo è più conciso e generico in quanto breve riassunto della segnalazione, il secondo è solitamente molto più articolato, dettagliato ed in grado di specificare maggiormente

il problema riscontrato dall'utente. Entrambe le caratteristiche sono fondamentali per la comprensione del testo da parte dei modelli e, per poterle racchiudere in una sola variabile testuale, sono state concatenate.

1.3.1 Dataset per Classificazione Binaria

Vengono considerati tre dataset distinti:

1. `Siderurgico`: composto da ticket di una celebre multinazionale italiana leader a livello mondiale nella produzione di impianti siderurgici;
2. `Multiservizi`: formato dai ticket di una nota azienda multiservizi italiana che fornisce servizi energetici, idrici e ambientali su suolo nazionale;
3. `Combinato`: costruito unendo i due dataset precedenti considerando ogni osservazione come fosse generata da un'unica istanza di HDA per non considerare l'effetto della specifica azienda sul testo.

Nella 1.1 vengono mostrate alcune caratteristiche di base dei dataset dopo esser stati suddivisi ininsieme di stima, verifica e convalida.

	Siderurgico	**Multiservizi**	**Combinato**
unità train set	9 173	16 452	25 625
unità validation set	3 670	6 582	10 251
unità test set	5 503	9 871	15 375
% incident train set	70.99%	13.15%	33.92%
token nel vocabolario	38 034	95 634	122 328
finestra temporale	8 mesi	10 mesi	10 mesi

Tabella 1.1: Sommario dataset per la classificazione binaria

1.3.2 Dataset per Classificazione Multiclasse

Anche per lo studio di classificazione multiclasse i clienti di riferimento sono due ma, in questo caso, non viene considerato anche un terzo dataset dato dall'unione dei primi due. Questa decisione è stata presa perchè i *problem*, a differenza degli *incident*, si assume possano avere natura totalmente diversa da un contesto all'altro. Unire *problem* di realtà differenti, infatti, porterebbe i modelli ad essere addestrati considerando una complessità spropositata ed inutile in quanto non è ipotizzabile un modello generale addestrato su *problem* riscontrati in diverse aziende. I due dataset considerati sono:

1. `Farmaceutico`: composto da ticket di una celebre multinazionale italiana che opera nell'area della salute e del benessere principalmente nel settore farmaceutico, ma anche nei prodotti di largo consumo, nella meccanica, nel vitivinicolo e nei profumi;

2. `Multiservizi`: i ticket considerati sono gli stessi del caso binario ma in questo caso la numerosità è inferiore in quanto son stati considerati solo i ticket aventi una categoria di CatalogFolder osservata almeno 35 volte nell'arco della finestra temporale considerata.

Come per i dataset precedenti vengono mostrate in tabella (1.2) alcune caratteristiche di base dei dataset dopo esser stati suddivisi in *train set*, *evaluation set* e *test set*.

	Farmaceutico	**Multiservizi**
unità train set	7205	3 937
unità validation set	1031	563
unità test set	2058	1 125
numero di Catalog Folder	50	43
token nel vocabolario	29 943	33 918
finestra temporale	7 mesi	10 mesi

Tabella 1.2: Sommario dataset per la classificazione multiclasse

Capitolo 2

Natural Language Processing e Large Language Model

L'intento di questo capitolo è quello di dare una definizione di *Natural Language Processing* (*NLP*), introducendo alcuni aspetti peculiari di questa branca del mondo della statistica e del *machine learning* percorrendone velocemente la storia recente evidenziando i continui passi avanti fatti menzionando gli approcci e le tecniche più comuni per l'elaborazione dei dati di tipo testuale. Riconoscendo nella Rete Neurale un modello fondamentale per questo studio, verrà spiegato poi nel dettaglio, dalla sua formulazione più semplice fino ad arrivare al *deep-learning* in quanto base di partenza per la costruzione dei *Large Language Model*. Infine, verranno illustrati in maniera approfondita la struttura e il funzionamento di uno dei più famosi e impattanti LLM: BERT il quale verrà poi utilizzato per la classificazione dei ticket.

2.1 NLP: Generalità e sviluppi recenti

Secondo Wikipedia (Wikipedia (2024)) la definizione di *Natural Language Processing* (*NLP*) è la seguente:

"L'elaborazione del linguaggio naturale (NLP, da natural language processing) è una sottobranca di linguistica, informatica e intelligenza artificiale che tratta l'interazione tra i computer e il linguaggio umano, in particolare sul come programmare i computer per elaborare e analizzare grande quantità di dati di linguaggio naturale. Lo scopo è rendere la tecnologia in grado di "comprendere" il contenuto dei documenti e le loro sfumature contestuali, in modo tale che possa quindi estrarre con precisione informazioni e idee contenute nei documenti, nonché classificare e categorizzare i documenti stessi."

Da tale definizione si comprende come NLP non sia solamente una specializzazione di un'unica materia ma piuttosto un campo interdisciplinare che combina statistica, linguistica e informatica per consentire al computer di comprendere, interpretare e generare linguaggio in modo quanto più simile a quello umano. Questo è un aspetto importante nel determinare la complessità e la varietà del tema.

Negli ultimi due decenni, l'Elaborazione del Linguaggio Naturale ha conosciuto un'evoluzione significativa, caratterizzata da importanti fasi di sviluppo. Dai primi anni 2000, con il crescere delle capacità computazionali e l'accumulo di enormi quantità di dati, il *machine learning* e il *deep learning* sono diventati sempre più centrali nel campo dell'NLP. Le Reti Neurali Ricorrenti o RNN (*Recurrent Neural Network*) (Pennington et al. (2014)), in particolare, sono state uno degli strumenti più utilizzati in questo campo. Questi modelli hanno offerto un approccio promettente per processare dati e catturare pattern locali nei dati testuali, rappresentando un passo avanti rispetto ai modelli tradizionali basati su regole o su metodi più semplici di rappresentazione delle parole come il *part-of-speech tagging*. Le RNN, infatti, sono progettate per gestire dati sequenziali come i dati testuali risultando in grado di catturare relazioni di breve termine tra le parole, come la struttura di frasi. Questa loro capacità era però piuttosto limitata in termini di memoria e il costo computazionale per l'addestramento di queste reti era piuttosto oneroso.

Tra il 2013 e il 2014, l'introduzione di vettori di parole come *Word2Vec* (Mikolov et al. (2013)) e *GloVe* (Pennington et al. (2014)) ha rivoluzionato l'approccio alla rappresentazione semantica delle parole, migliorando la comprensione del significato delle parole in base al contesto. Queste tecniche hanno permesso di superare la classica rappresentazione "*one-hot*", dove ogni testo veniva codificato come un vettore binario di dimensioni pari al numero di parole che compongono il vocabolario. L'idea alla base di queste nuove rappresentazioni è quella di codificare i testi sotto forma di vettori densi in uno spazio di dimensione molto più piccola rispetto a quella del vocabolario del *corpus* in modo tale da fornire come input per i modelli predittivi costrutti più informativi possibili. Il principio alla base del modello *Word2Vec* è quella di allenare reti neurali a codificare testi in vettori tali che, per parole simili, siano collocati in spazi vettoriali simili. Questo modello è noto per la sua efficienza computazionale che lo rende adatto a grandi quantità di dati, e per la sua capacità di produrre rappresentazioni semanticamente significative. Esso però, essendo basato sull'addestramento di reti neurali, per portare buoni risultati, necessita sempre di grandi quantità dei dati, ed ha la tendenza ad ignorare relazioni globali tra le parole in un *corpus*. *GloVe*, per apprendere le

relazioni, non sfrutta reti neurali ma statistiche globali del *corpus* come le frequenze di co-occorrenza delle parole. Al contrario del *Word2Vec* infatti, per costruzione, è in grado di catturare relazioni globali tra le parole ed è meno dipendente dalla mole di dati risultando più indicato per *corpus* di dimensione ridotta ma soffre di una maggiore complessità computazionale.

Una pietra miliare nella storia evolutiva dell'elaborazione del linguaggio naturale è stata la pubblicazione nel 2017 dell'articolo *"Attention is All You Need"* di A. Vaswani (Vaswani et al. (2017)) che introduce l'architettura *Transformer* basata su meccanismi di attenzione. Il *paper* introduce il concetto di *self-attention*: un meccanismo che permette al modello di assegnare pesi differenti alle parole all'interno di una sequenza (come una frase) in base alle loro relazioni semantiche. Questo meccanismo è cruciale per catturare le dipendenze a lungo termine e comprendere il contesto delle parole in modo più efficace. Un altro punto di forza chiave di questo modello è il fatto che rende possibile la parallelizzazione efficiente durante l'addestramento, eliminando la necessità di elaborare le parole in modo sequenziale ed abbattendo i limiti dati dal costo computazionale caratteristico degli approcci precedenti.

Nel corso del 2018 si è fatto sempre più evidente l'utilizzo di un nuovo approccio: quello basato sul pre-addestramento seguito da *fine-tuning*. Modelli come BERT (*Bidirectional Encoder Representations from Transformers*) (Devlin et al. (2018)), sviluppato dai ricercatori di Google, e GPT (*Generative Pre-trained Transformer*) (Radford et al. (2018)), sviluppato da OpenAI, hanno mostrato prestazioni eccezionali in una grande varietà di applicazioni NLP. Questo approccio consiste, in prima battuta, nell'addestramento supervisionato di un modello generico di linguaggio su un ampio *corpus* di testi chiamato *Large Language Model* (LLM). Questo processo permette al modello di apprendere rappresentazioni linguistiche e di catturare il contesto generale e la semantica delle parole per poi essere adattato al caso specifico in analisi attraverso la tecnica del *fine-tuning*. Tale procedura consiste semplicemente in un ulteriore addestramento supervisionato del modello utilizzando come dati quelli specifici del problema in esame in modo tale da specializzarlo.

La storia successiva delle ricerche in materia NLP, fino ad arrivare ai giorni nostri, è caratterizzata dalla continua evoluzione degli LLM. Sono stati sviluppati modelli multilingue come *mBERT* addestrati su testi multilingua. Tale addestramento porta *mBERT* a godere della proprietà *"zero-shot learning"*, ovvero, applicato al contesto dell'apprendimento delle lingue, la capacità del modello di applicare le conoscenze acquisite da una lingua ad un'altra senza la necessità di un addestramento specifico per la specifica lingua di interesse. Ciò porta il modello ad essere estremamente utile per compiti che

coinvolgono diverse lingue, e capace di fornire una rappresentazione globalmente condivisa dello spazio semantico del linguaggio naturale.

L'evoluzione degli *Large Language Models* è proseguita anche in termini di grandezza e complessità grazie alla crescita della potenza di calcolo disponibile. Uno dei passi significativi in questo percorso è stato il lancio dal modello GPT-3 (*Generative Pre-trained Transformer 3*) di OpenAI (Brown et al. (2020)), rilasciato nel giugno 2020. GPT-3 ha segnato un notevole salto in termini di dimensioni ed ha riscosso un grande successo dimostrando la capacità dei LLM di eseguire un'ampia varietà di compiti di NLP e di complessità sampre maggiore.

L'aumento delle dimensioni e della potenza dei modelli, tuttavia, ha portato con sé preoccupazioni riguardanti i costi computazionali e la concentrazione di risorse in pochi attori con accesso alle infrastrutture necessarie. È cresciuta, e continua a farlo tuttora, anche la consapevolezza riguardo alle questioni etiche e di *bias* nell'NLP. L'uso di modelli di linguaggio addestrati su enormi quantità di testi, infatti, può portare alla replicazioni di pregiudizi esistenti e all'amplificazione di stereotipi con conseguenze molto gravi in alcune applicazioni. Quest'ultimo tema è al centro dell'attenzione sia della comunità di ricerca che delle aziende del settore: entrambe, infatti, si sono impegnate a indagare e ad affrontare queste problematiche sviluppando linee guida per la creazione di modelli etici e trasparenti. Ne è un esempio molto recente l'*AI Act* (Parlamento Europeo (2023)) approvato dalla Commissione Europea il 9 dicembre 2023.

2.2 Dalla Rete Neurale One-Layer al Deep Learning

Come anticipato nella sezione precedente, l'impatto delle Reti Neurali e loro derivazioni sui problemi in ambito NLP è molto forte. In particolare esse sono la base della formulazione dei *Large Language Model* e ricoprono un ruolo cruciale in questo studio. Per questa ragione si vuole per mezzo di questa sezione descrivere i fondamenti di tale modello, dalla loro versione più semplice, ovvero la Rete Neurale ad un unico strato latente (*One-Layer*), fino ad estensioni molto più articolate come il *Transformer*.

Una generica Rete Neurale ad unico strato latente può essere vista come uno schema di regressione a due stadi di cui almeno uno non lineare. Possiamo immaginarla come un grafo aciclico in cui sono presenti tre strati di nodi (*input*, *hidden* e *output*) collegati tra loro sequenzialmente attraverso archi pesati. Dato il nodo di input x_h, il nodo nascosto

z_j e il nodo di output y_k, la formulazione generale del modello può essere la seguente:

$$z_j = f_0\left(\sum_{h\to j}(\alpha_{hj}x_h + b_j)\right), \quad y_k = f_1\left(\sum_{j\to k}(\beta_{jk}z_j + b_k)\right) \tag{2.1}$$

dove f_0 e f_1 sono opportune funzioni dette *funzioni di attivazione*, mentre gli α_{hj} e β_{jk} sono parametri che quantificano il peso delle relazioni (archi) che intercorrono tra i nodi. I termini b_j e b_k sono detti *bias* e hanno la funzione di dare peso ai nodi indipendentemente dall'input che ricevono. I pesi e i *bias* sono parametri da stimare in modo tale da ottimizzare una certa funzione obiettivo.

Un'importante proprietà di cui gode questo modello è il fatto di essere un approssimatore universale. Si può dimostrare, infatti, che una rete neurale con un singolo strato nascosto può approssimare qualsiasi funzione continua su un intervallo compatto mediante la combinazione lineare di un numero sufficientemente grande di funzioni di base. In altre parole, con una giusta scelta dei pesi e di *bias* della rete, è possibile approssimare qualsiasi funzione desiderata (come la funzione di probabilità della variabile y) con una precisione arbitraria.

L'algoritmo di addestramento delle reti neurali più diffuso è il *Back-propagation*: un algoritmo iterativo che, come suggerisce il nome, si basa sul concetto di propagazione all'indietro dell'errore nella rete. La strategia è quella di minimizzare l'errore di previsione minimizzando una certa funzione di costo (*Loss function*) che indichiamo in maniera generica con $L(\theta)$, con θ vettore dei parametri della rete. Gli step che compongono l'algoritmo di stima sono i seguenti:

1. **Inizializzazione dei pesi**: alla prima iterazione i parametri $\theta^{(0)}$ vengono inizializzati in maniera casuale o al più secondo una distribuzione specifica particolarmente vantaggiosa

2. **Propagazione in avanti (*Feedforward Pass*)**: dalla seconda iterazione in poi, ad ogni t-esima iterazione, per ciascuna osservazione usata per l'addestramento, l'input x viene propagato attraverso la rete neurale utilizzando i pesi correnti ricavando il valore dei nodi nascosti e di quelli di output

$$z_j^{(t)} = f_0\left(\sum_{h\to j}(\alpha_{hj}^{(t-1)}x_h^{(t)} + b_j^{(t-1)})\right), \quad y_k^{(t)} = f_1\left(\sum_{j\to k}(\beta_{jk}^{(t-1)}z_j^{(t)} + b_k^{(t-1)})\right)$$

3. **Calcolo dell'errore**: viene quantificato l'errore confrontando l'output generato

dalla rete con i pesi correnti mediante il calcolo della funzione di perdita

$$e^{(t)} = L\left(\theta^{(t)}\right)$$

4. **Propagazione all'indietro (*Backward Pass*)**: l'errore appena calcolato viene propagato all'indietro calcolando i gradienti della funzione di costo rispetto ai pesi e ai *bias* della rete, utilizzando la *regola della catena*. Per i generici pesi α_{hj} e β_{jk} e i generici *bias* b_j e b_k i gradienti vengono calcolati come:

$$\begin{aligned}
\frac{\partial e^{(t)}}{\partial \alpha_{hj}^{(t-1)}} &= \frac{\partial e^{(t)}}{\partial z_j^{(t)}} \cdot \frac{\partial z_j^{(t)}}{\partial \alpha_{hj}^{(t-1)}} \\
\frac{\partial e^{(t)}}{\partial \beta_{jk}^{(t-1)}} &= \frac{\partial e^{(t)}}{\partial y_k^{(t)}} \cdot \frac{\partial y_k^{(t)}}{\partial \beta_{jk}^{(t-1)}} \\
\frac{\partial e^{(t)}}{\partial b_j^{(t-1)}} &= \frac{\partial e^{(t)}}{\partial z_j^{(t)}} \cdot \frac{\partial z_j^{(t)}}{\partial b_j^{(t-1)}} \\
\frac{\partial e^{(t)}}{\partial b_k^{(t-1)}} &= \frac{\partial e^{(t)}}{\partial y_k^{(t)}} \cdot \frac{\partial y_k^{(t)}}{\partial b_k^{(t-1)}}
\end{aligned}$$

5. **Aggiornamento dei pesi**: dopo aver calcolato i gradienti, i pesi e i *bias* vengono aggiornati nella direzione opposta del gradiente della funzione di costo appena calcolato. Il peso dell'aggiornamento che avviene ad ogni iterazione viene modulato attraverso l'importante parametro di regolazione *learning rate* η. I generici parametri aggiornati della rete vengono calcolati come:

$$\begin{aligned}
\alpha_{hj}^{(t)} &= \alpha_{hj}^{(t-1)} - \eta \frac{\partial e^{(t)}}{\partial \alpha_{hj}^{(t-1)}} \\
\beta_{jk}^{(t)} &= \beta_{jk}^{(t-1)} - \eta \frac{\partial e^{(t)}}{\partial \beta_{jk}^{(t-1)}} \\
b_j^{(t)} &= b_j^{(t-1)} - \eta \frac{\partial e^{(t)}}{\partial b_j^{(t-1)}} \\
b_k^{(t)} &= b_k^{(t-1)} - \eta \frac{\partial e^{(t)}}{\partial b_k^{(t-1)}}
\end{aligned}$$

L'algoritmo procede iterando dal secondo step all'ultimo un numero predeterminato di iterazioni, in questo contesto dette *epoche*, o fino al raggiungimento di un criterio di convergenza. Durante ciascuna iterazione, vengono eseguiti i passaggi di forward pass, calcolo dell'errore, backward pass, aggiornamento dei pesi e dei bias, e valutazione delle

prestazioni. L'iterazione termina quando l'errore della rete non diminuisce più significativamente o quando viene raggiunto il numero massimo di epoche.

Considerando il fatto che ogni strato latente può avere un gran numero di nodi e possono esserci più strati latenti, il numero di parametri del modello può raggiungere cifre elevatissime. Calcolare il gradiente di ciascun parametro ad ogni epoca considerando tutte le osservazioni può portare quindi a dei costi computazionali proibitivi sia in termini di velocità che di memoria. Per poter mitigare questo aspetto, la tecnica comunemente utilizzata è quella del *Mini-Batch*, la cui idea di base è quella di gestire l'elaborazione dei dati in porzioni più piccole, anziché elaborare l'intero set di dati di addestramento ad ogni epoca. Con questa procedura il set di addestramento viene quindi suddiviso in piccoli sottoinsiemi chiamati *mini-batch* di dimensione B costante (solitamente 32, 64 o 128) e si eseguono il *Feedforward* e il *Backward* su ciascuno di essi separatamente. Quindi, viene calcolato il gradiente dell'errore solo per quel *mini-batch* specifico. Questo processo viene ripetuto per tutti i *mini-batch* costituendo un'epoca di addestramento della rete. L'impiego di questa procedura gestisce il problema della velocità di calcolo in quanto ogni epoca può essere facilmente parallelizzata velocizzando in maniera esponenziale il processo di stima. Anche a livello di memoria l'impiego di *mini-batch* è molto vantaggioso perché è possibile considerare dataset di grandi dimensioni senza doverli caricare interamente in memoria. Un altro aspetto importante è che la casualità della composizione nei *mini-batch* introduce del rumore durante l'aggiornamento dei pesi, il che può aiutare a prevenire il sovra-adattamento. Il numero B di osservazioni che compongono ciascun *mini-batch* è un iperparametro importante che deve essere scelto principalmente in base alla memoria disponibile.

Per quanto riguarda la funzione di costo da minimizzare, essa è specifica della natura del problema che si sta affrontando. Vengono presentate di seguito quelle che effettivamente vengono utilizzate nello studio, ovvero le più comuni nell'ambito della classificazione binaria e dalla classificazione multiclasse.

Per lo scenario multiclasse la *Loss-Function* considerata è la Cross-Entropia (H_{CE}) che mira a misurare la differenza tra la distribuzione di probabilità delle K classi predette $\hat{y}$ e la distribuzione di probabilità di quelle reali y:

$$\begin{aligned} H_{CE}(\hat{y}, y) &= -\sum_{k=1}^{K} y_k \, log(\hat{y}_k) \\ &= -\sum_{k=1}^{K} \sum_{i=1}^{n} y_{ki} \log(\hat{y}_{ki}) \end{aligned} \tag{2.2}$$

Dove y_{ki} è la probabilità reale della k-esima classe per l'osservazione i nell'insieme di stima e $\hat{y}_{ki}$ quella stimata dal modello. Ciascun termine $y_{ki}\, log(\hat{y}_{ki})$ rappresenta la perdita per la singola classe e la loro somma produce la Cross-Entropia totale.

Nel contesto binario, invece, viene utilizzata una specificazione della BC chiamata Cross-Entropia Binaria (H_{BCE}) calcolata come:

$$H_{BCE}(\hat{y}, y) = -\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{2.3}$$

Dove y_i e $\hat{y}_i$ sono rispettivamente la probabilità reale e stimata per l'i-esima osservazione di appartenere alla classe di riferimento.

Come la funzione di perdita, anche le funzioni di attivazione f_0 e f_1 sono elementi fondamentali nell'architettura di una rete neurale, anche in questo caso ne esistono molte e vengono scelte in base al contesto in cui ci si trova e a loro specifiche proprietà. Come introdotto nella descrizione del *Backpropagation*, a ciascun strato che compone la rete viene associata una funzione di attivazione con lo scopo di definire l'output di ciascun nodo a partire dall'insieme dei suoi input. In pratica, ogni nodo riceve dallo strato precedente gli input, li moltiplica ciascuno per il relativo peso, li somma e a questa somma applica la funzione di attivazione. Le funzioni di attivazione devono essere monotone e derivabili e, a seconda della situazione, possono dover godere di alcune proprietà (ad esempio avere supporto definito in uno specifico intervallo). Un'ulteriore proprietà richiesta alla maggioranza di queste funzioni è la capacità di introdurre la non linearità nella rete, portandola ad essere in grado di "catturare" relazioni non lineari nell'insieme dei dati. Di seguito vengono presentate le funzioni di attivazione più diffuse, evidenziando i loro tratti distintivi e la modalità del loro impiego all'interno della rete.

- **Lineare**: anche detta funzione identità, è definita come:

  $$f(x) = x$$

  Ha supporto in tutto l'asse reale ed è utilizzata per rilevare strutture lineari nei dati. Tuttavia il suo gradiente è costante perciò, durante la retropropagazione del gradiente, può portare ad una limitazione della capacità di apprendimento del modello e alla saturazione del gradiente.

- **Sigmoide**: è definita come:

  $$f(x) = \frac{1}{1 + e^{-x}}$$

Riceve in input un numero reale e lo mappa tra 0 e 1. È non lineare, differenziabile e monotona e viene spesso usata per lo strato d'output nei problemi di classificazione binaria in quanto il suo risultato può essere interpretato come la probabilità di appartenenza alla classe di riferimento. È computazionalmente onerosa e non centrata in 0.

- **Tangente iperbolica**: la sua formulazione è:

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Essendo una modifica della funzione sigmoide ha caratteristiche simili. La tangente iperbolica, però, mappa qualsiasi valore reale in un valore compreso tra –1 e 1, è centrata sullo 0, monotona e differenziabile ovunque. Può essere utilizzata negli strati nascosti quando si vuole avere una contrazione verso lo 0 degli output dei nodi nascosti oppure come strato d'output in problemi di regressione quando si è interessati a previsioni in scala simmetrica rispetto allo 0.

- **ReLU (Unità Lineare Rettificata)**: la sua definizione è:

$$f(x) = max\{0,\, x\}$$

È definita nell'intervallo $[0, +\infty)$, è monotona così come la sua derivata. È molto utilizzata come funzione di attivazione perché risolve il problema detto "gradiente evanescente" per cui il gradiente, propagandosi nella rete, diminuisce gradualmente. Essa però soffre del problema chiamato "ReLu morente" in quanto i nodi con valori negativi avranno gradiente nullo e quindi non verranno più ottimizzati. Questa funzione di attivazione viene usata solamente all'interno degli strati nascosti, non d'output appunto perché, in caso di valori negativi si avrebbe la "morte" del nodo. Tale condizione è accettabile negli strati nascosti in quanto induce la sparsità dei parametri e migliora l'efficienza dell'addestramento della rete.

- **Softmax**: la sua definizione è la seguente:

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Può essere descritta come la combinazione di molteplici funzioni sigmoidali. La funzione Softmax restituisce la probabilità per un'osservazione di appartenere a ogni singola classe. Essa restituisce quindi un vettore di dimensione pari al numero di categorie K di valori compresi nell'intervallo [0, 1] che sommano a 1.

Viene utilizzata come funzione di attivazione dello strato d'output nei problemi di classificazione multiclasse.

Rispetto alle reti neurali con un unico strato latente, le reti neurali profonde (*Deep Neural Networks*, DNN) rappresentano una prima evoluzione significativa che consiste principalmente nell'aggiunta di più strati nascosti tra quelli di input e di output, permettendo alla rete di apprendere rappresentazioni più complesse e di catturare pattern non lineari più sofisticati nei dati.

Il termine *Deep Learning* si riferisce proprio all'uso di reti neurali con molti strati nascosti, spesso decine o addirittura centinaia. L'aumento della profondità della rete utilizzando un'ampia varietà di funzioni di attivazione permette di rappresentare funzioni molto complesse, ma introduce anche nuove sfide computazionali che rendono più complesso l'addestramento di reti molto profonde tra cui il problema del gradiente evanescente su tutti, il cui effetto in questo contesto risulta molto più marcato. Per mitigare l'impatto del gradiente evanescente, vengono largamente utilizzate, quindi, funzioni di attivazione avanzate come la ReLU definita in precedenza e varianti (*Leaky* ReLU, *Parametric* ReLU, ELU) le quali aiutano a mantenere i gradienti significativi durante il *Backpropagation*.

Con l'evoluzione del *deep learning*, sono state sviluppate diverse architetture più complesse per affrontare problemi specifici. Le più diffuse sono le Reti Neurali Convoluzionali (CNN) per l'elaborazione di immagini, le Reti Neurali Ricorrenti (RNN) per l'elaborazione di sequenze già citate in precedenza, e i *Transformers* per catturare dipendenze a lungo termine. Per le prime due tipologie viene fornita una breve introduzione di seguito, la terza, invece, viene approfondita più nel dettaglio nella Sezione 2.3 dove viene spiegato il modello di riferimento di questo studio: BERT.

Le CNN sono particolarmente adatte per il processamento di dati con struttura spaziale, come immagini e video. Una CNN è composta da strati convoluzionali che applicano filtri convoluzionali all'input, seguiti da strati detti di *pooling* che ne riducono la dimensionalità. Possiamo infatti vedere uno strato di *pooling* come l'applicazione di una funzione di aggregazione come, ad esempio, la media all'output del filtro convoluzionale. La trasformazione convoluzionale per un filtro k-esimo nello strato l può essere formulata come:

$$h_k^{(l)} = f\left(\sum_{c=1}^{C^{(l-1)}} \left(W_{kc}^{(l)} * h_c^{(l-1)}\right) + b_k^{(l)}\right)$$

dove $*$ indica l'operazione di convoluzione, $C^{(l-1)}$ è il numero di output dello strato precedente, e $W_{kc}^{(l)}$ e $b_k^{(l)}$ sono rispettivamente i pesi e il *bias* del filtro k-esimo.

Le RNN, invece, sono progettate per processare dati di struttura sequenziale e l'idea

alla loro base è quella di mantenere uno stato interno h_t che viene combinato. Esso viene aggiornato ad ogni step temporale t e cattura informazioni sull'input ricevuto fino a quel punto della sequenza di dati, rappresentando la memoria della rete in un dato momento t. La formulazione generale di una RNN nello strato l è:

$$h_t^{(l)} = f_h\left(W_h^{(l)} h_{t-1}^{(l)} + W_x^{(l)} x_t + b_h^{(l)}\right)$$

dove $h_t^{(l)}$ è lo stato nascosto al tempo t, $W_h^{(l)}$ e $W_x^{(l)}$ sono i pesi della connessione rispettivamente dello stato precedente e dell'input corrente, e $b_h^{(l)}$ è il *bias*.

Le reti neurali sono un metodo di *machine-learning* particolarmente soggetto al problema del sovra-adattamento e, all'aumentare della loro complessità, questa loro propensione è sempre più marcata. Esse infatti, a causa del numero di parametri tipicamente molto elevato e alla loro grande elasticità, hanno la tendenza ad adattarsi fin troppo bene ai dati di addestramento perdendo così la capacità di generalizzazione sui nuovi dati con una conseguente limitazione in termini di capacità predittiva. Per questa ragione esistono numerose tecniche che possono essere usate e/o combinate per arginare questo problema. Quelle più comunemente usate sono:

- **Convalida incrociata**: tecnica non specifica del caso delle reti neurali, viene utilizzata molto in qualsiasi contesto che prevede la stima di un modello predittivo, specialmente nelle fasi di selezione degli iperparametri ottimali dei modelli ed è particolarmente indicata quando il numero di osservazioni per la stima dei modelli è bassa rispetto al numero di parametri da stimare. Essa consiste nel partizionare l'insieme di stima in k sotto insiemi detti "*fold*". Di questi, $k-1$ vengono usati per la stima ed il restante per la valutazione delle prestazioni del modello. Iterativamente ciascun *fold* viene considerato come set di convalida fino a giungere al punto in cui son state considerate tutte le k possibili configurazioni. In questo modo i parametri della rete vengono sempre aggiornati e valutati su dati diversi aumentando la capacità di generalizzazione.

- **Regolarizzazione L1 e L2**: consiste nell'aggiunta di un termine di penalità che dipende dai parametri alla funzione di perdita. Le tipologie di penalizzazione più diffuse sono la regolarizzazione L1 e L2. In letteratura, nel contesto delle reti neurali, la penalità L2 viene spesso indicata come "*weight decay*". Per una trattazione più esaustiva di queste procedure si rimanda all'Appendice A.

- **Dropout**: è una tecnica di regolarizzazione specifica delle reti neurali che prevede che ad ogni epoca di addestramento alcuni nodi degli strati nascosti scelti in

maniera casuale vengano "disattivati" (impostati a 0 a prescindere dalla funzione di attivazione e dall'input). Questo impedisce alla rete di dipendere troppo da singoli nodi e promuove una rappresentazione più robusta inserendo aleatorietà nel processo, portando notevoli miglioramenti in termini di sovra-adattamento.

In generale, selezionare le tecniche ideali e gli iperparametri ottimali nel contesto delle reti neurali è un processo molto lungo e complicato. La varietà delle strade da poter percorrere è molto ampia e non esistono regole pratiche che stabiliscano direttamente configurazioni preferibili.
Principalmente le scelte da effettuare prevedono la selezione dell'architettura in base al problema considerato, del numero di strati latenti, del numero di nodi per ogni strato latente, del *learning rate* η, delle funzioni di attivazione con relativi parametri, della dimensione da considerare nei mini-batch, del numero di epoche di addestramento, se inserire o meno la regolarizzazione L1 o L2, se utilizzare la tecnica del dropout e, nel caso, stabilire la proporzione di nodi da porre a 0.

2.3 BERT: Bidirectional Encoder Representations from Transformers

In questa sezione viene approfondito in modo preciso ed accurato il modello BERT spiegando la sua architettura e procedura di stima descritta in Devlin et al. (2018) in cui viene introdotto per la prima volta. La scelta di dedicare una sezione intera alla descrizione di questo modello specifico è dettata dal fatto che esso gioca un ruolo fondamentale nell'analisi che verrà condotta poichè da esso derivano i *Large Language Model* preaddestrati utilizzati.

Come introdotto nella sezione precedente, BERT (*Bidirectional Encoder Representations from Transformers*) è un potente modello di rappresentazione del linguaggio che ha rivoluzionato il mondo NLP. Sviluppato da Google nel 2018, esso è un'architettura basata su *deep neural network* pre-addestrata su enormi quantità di dati di testo e può essere ottimizzato per varie attività di NLP.

L'architettura del modello si basa su un codificatore *Transformer* bidirezionale multistrato. Nello specifico, la prima versione di BERT ($\text{BERT}_{\text{BASE}}$) utilizza $L = 12$ strati di *Transformer*, e $H = 768$ nodi nascosti, per un totale di circa 110 milioni di parametri. A differenza dei modelli linguistici tradizionali pensati per elaborare il testo da sinistra a destra o da destra a sinistra, BERT è un modello bidirezionale che legge l'intera sequenza di parole in input di una frase in entrambe le direzioni in ognuno dei suoi strati,

potenziando significativamente la capacità di catturare il contesto e il significato delle parole in modo più accurato. Dato che il modello è riducibile a una composizione di strati *Tranformer*, per poter spiegare come avviene l'elaborazione dei dati testuali dati in input a BERT è necessario approfondire questa particolare architettura in tutte le sue componenti.

Transformer è una particolare rete neurale complessa con una struttura codificatore-decodificatore (*decoder-encoder*) (Figura 2.1), dove una sequenza testuale viene data in input al codificatore che la mappa in rappresentazioni continue, mentre il decodificatore genera una sequenza di output un elemento alla volta.

- **Encoder**: è composto da sei strati identici, ciascuno di essi a sua volta è composto da due componenti principali (sotto-strati), il *Multi-Head Self-Attention Mechanism* e una più semplice rete neurale *Feed-Forward*. L'output di ciascun sotto-strato viene normalizzato.

- **Decoder**: anch'esso è composto da sei strati identici ma, oltre agli stessi due componenti dell'*encoder*, contiene un terzo sottostrato che altro non è che un ulteriore *Multi-Head Self-Attention Mechanism* applicato all'output finale del *decoder*. Quest'ultimo substrato ha la peculiarità di avere la mascheratura delle posizioni successive alla $i - esima$ assicurando che le previsioni per la posizione i possano dipendere solo dagli output conosciuti nelle posizioni precedenti a i.

Il Meccanismo di Attenzione (*Attention Mechanism*) è quello che permette al modello di caratterizzare la somiglianza tra diversi elementi nella stessa sequenza. Il modello in questo modo riesce a capire quali parti della frase sono maggiormente legate, catturando così le dipendenze tra esse con un conseguente miglioramento della comprensione del contesto. Un meccanismo di attenzione può essere descritto come la mappatura di una *query* e di un insieme di coppie *chiave-valore* su un output), dove la *query*, le *chiavi*, i *valori* e l'output sono dei vettori. Lo *score* di attenzione viene calcolato come somma pesata dei *valori*, dove il peso assegnato a ciascun *valore* viene calcolato da una funzione di compatibilità della *query* con la *chiave* corrispondente. Il *Multi-Head self-attention mechanism* non è altro che l'unione di più meccanismi attenzione eseguiti in parallelo. L'attenzione utilizzata nel paper viene definita "Prodotto Scalare Scalato" (*Scaled Dot-Product Attention*). Siano i vettori delle *query* e delle *chiavi* di dimensione d_k e quello dei *valori* di dimensione d_v. Nella pratica la funzione di attenzione viene calcolata simultaneamente su un insieme di *query*, raggruppate in una matrice Q. Anche le *chiavi* e i *valori* sono raggruppati nelle matrici K e V. La matrice degli *output* di una singola

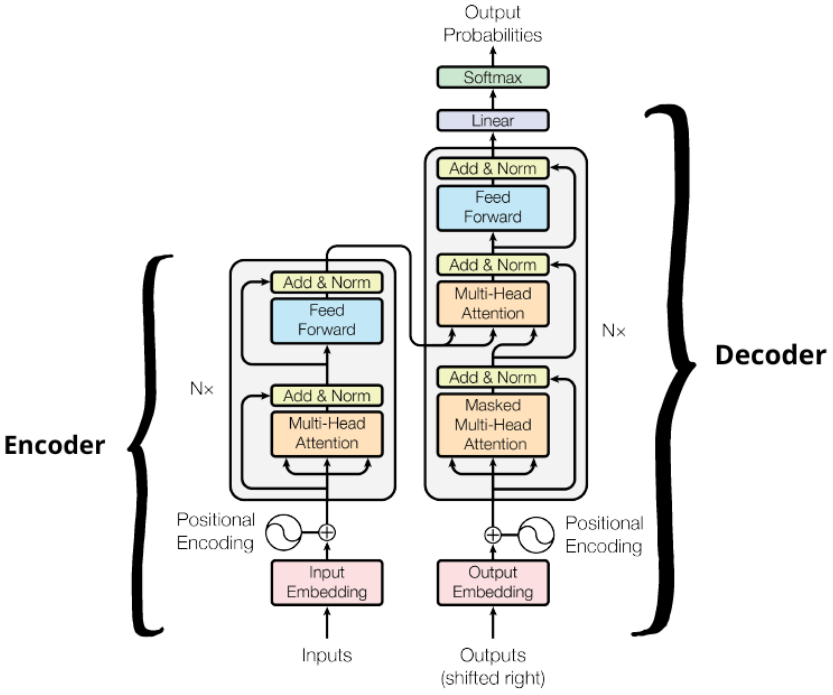

FIGURA 2.1: Architettura del modello Transformer

funzione di attenzione viene quindi calcolata come:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

L'estensione della formula 2.4 al caso *Multi-head* viene ottenuta proiettando linearmente *query*, *chiavi* e *valori* h-volte con proiezioni diverse, rispettivamente di dimensioni d_k, d_k e d_v. Su ciascuna di queste versioni proiettate viene calcolata la funzione di attenzione in parallelo ottenendo output d_v-dimensionali. Questi vengono quindi concatenati e ancora una volta proiettati, arrivando alla definizione della formula:

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, \ldots, head_h)W^O \\ dove\ head_i &= Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.5}$$

dove le proiezioni sono matrici di parametri $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ e $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. Nello specifico, nel caso di BERT, viene posto $h = 8$ e $d_k = d_v = d_{model}/h = 64$. Si può osservare la rappresentazione del calcolo della singola funzione di attenzione e del meccanismo *Multi-Head Attention* in Figura 2.2.

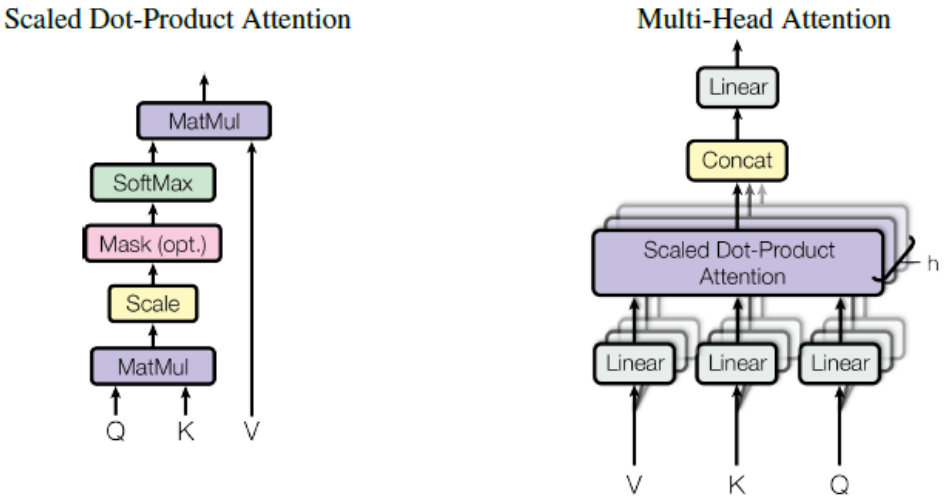

FIGURA 2.2: Scaled Dot-Product Attention (sx) e Multi-Head Attention (dx)

Il secondo sotto-strato è la rete *Feed-Forward* (FFN) completamente connessa, impiegata successivamente ai meccanismi d'attenzione dell'*encoder* e del *decoder*. Essa viene applicata separatamente e con pesi costanti a ciascun elemento del vettore di input e si compone di due trasformazioni lineari, intervallate dalla funzione d'attivazione ReLU in cui lo strato nascosto ha $d_{ff} = 2048$ nodi mentre le sequenze in ingresso e in uscita sono di dimensione $d_{\text{model}} = 512$:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.6}$$

Infine, il termine di *bias* posizionale volto ad aggiungere informazione sul rango dei token nella squenza che viene utilizzato è quello sinusoidale (*Sinusoidal Positional encoding*). Esso è un metodo comune nelle reti neurali trasformative che utilizza funzioni sinusoidali per generare una codifica di posizione per ciascun token e, nel caso dell'architettura *Transformers*, viene applicato prima di fornire l'input al meccanismo di multi-attenzione. Tornando all'applicazione specifica dei *Transformers* su BERT, l'input fornito all'architettura non è il testo grezzo, ma una sua trasformazione preliminare. Per rendere il modello in grado di poter gestire le varie applicazioni in ambito NLP, infatti, ciascun testo assume una rappresentazione che non genera ambiguità, sia il testo formato da una frase corta, articolata o addirittura da più frasi (es: domanda-risposta), chiamata *Token Embeddings*. Ciascun testo viene in prima battuta tokenizzato dall'algoritmo di tokenizzazione WordPiece (Wu et al. (2016)), dove con tokenizzazione si intende in generale la procedura di scomposizione del singolo testo in una sequenza di singole unità significative (*token*) le quali solitamente sono le parole che lo compongono, ma possono essere anche frasi, simboli, o qualsiasi altra componente rilevante. Ai

token così formati ne vengono aggiunti altri di supporto: un token di classificazione *[CLS]* nella prima posizione di ogni sequenza e un token di separazione tra le frasi nella sequenza *[SEP]*. Oltre gli *embeddings* di tokenizzazione vengono aggiunti anche quelli di segmentazione che permettono di assegnare un'etichetta marcando le diverse frasi nella sequenza e quelli di posizione. Viene mostrata in Figura 2.3 la composizione della rappresentazione di una frase d'esempio nel caso in cui i testi siano formati da coppie di frasi (A e B).

Per rendere BERT un *Large Language Model* a tutti gli effetti che contenga infor-

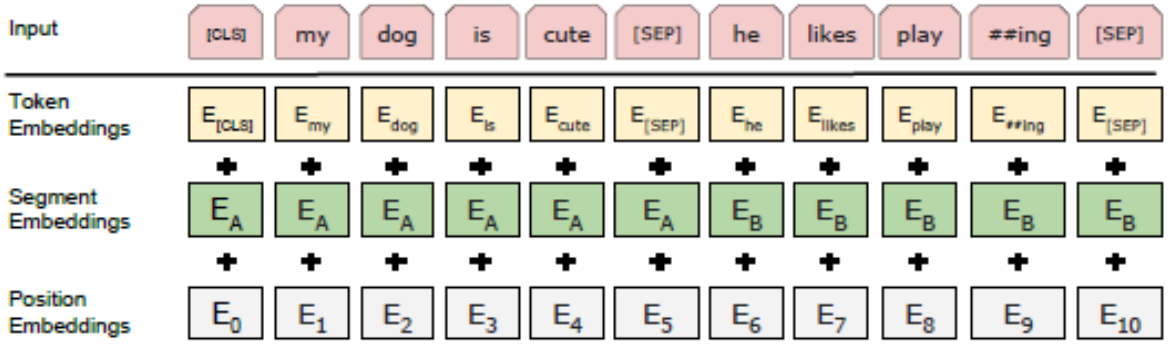

FIGURA 2.3: Esempio di rappresentazione di input.

mazioni pregresse sulle relazioni semantiche e contestuali delle frasi e/o singole parole, cruciale importanza assume il suo pre-addestramento. Questo avviene ottimizzando i parametri del modello utilizzando due task non supervisionati defeniti come *Masked Language Modeling* (MLM) e *Next Sentence Prediction* (NSP). Il primo compito di BERT è quello di prevedere i valori originali di token mascherati dal token *MASK* in base al loro contesto. Per questo addestramento viene mascherato in maniera casuale il 15% dei token di input. Il task NSP consente invece a BERT di comprendere la relazione tra due frasi prevedendo se due frasi sono consecutive o meno in un dato testo. Queste due tipologie di pre-addestramento non supervisionato vengono eseguite sfruttando un enorme corpus di testi portando BERT ad "imparare" in maniera vera e propria relazioni semantiche generali del linguaggio.

La conoscenza così maturata, pur essendo estremamente ricca e vasta, risulta però spesso troppo generica per affrontare compiti specifici nel campo dell'NLP. Ogni applicazione pratica, infatti, richiede la capacità di catturare e comprendere pattern peculiari e dettagliati del dominio specifico. Per colmare questa lacuna, viene impiegata la tecnica del *fine-tuning* brevemente descritta nella Sezione 2.1, che consente di adattare BERT alle esigenze particolari di un determinato problema. Il processo di *fine-tuning*

prevede l'ulteriore reiterazione dell'algoritmo di stima del modello su un set di dati specifico, permettendo di ottimizzare i suoi parametri per migliorare le prestazioni su tale task. Considerando l'esempio effettivamente attuato in questo studio, nei problemi di classificazione di testi, il fine-tuning comporta l'aggiunta di uno strato di classificazione finale al modello BERT pre-addestrato. Questo strato aggiuntivo è specificamente progettato per il compito di classificazione, e il modello completo, comprensivo dello strato di classificazione, viene addestrato su un dataset etichettato. In questo modo si arriva ad ottenere un modello altamente specializzato e ottimizzato, capace di svolgere con maggiore accuratezza e efficienza il compito per cui è stato fine-tuned.

Capitolo 3

Trasformazione dei testi

L'intento di questo capitolo è quello di definire in maniera puntuale le difficoltà analitiche e strutturali che portano con sé questa tipologia di dati, fornendo alcuni esempi di procedure comuni applicate in questi casi.

3.1 Dati Testuali

Nell'ambito dell'analisi statistica e *machine learning* tradizionale i dati a disposizione sono numerici e strutturati in formato tabellare in cui a ciascuna unità statistica viene dedicata una riga e a ciascuna variabile osservata viene dedicata una colonna. Tendenzialmente quindi, salvo la presenza di valori mancanti o particolari eccezioni, per ogni unità vengono registrate le stesse variabili in modo tale che l'input di qualsivoglia modello di previsione abbia dimensione costante. Questa struttura non è così immediata da costituire quando si ha a disposizione dataset, come nel caso in esame, in cui si dispone di una sola variabile esplicativa testuale ed una variabile dipendente y da prevedere sfruttando la covariata testuale.

Per poter quindi analizzare ed elaborare dati di questo tipo è necessario trovare una rappresentazione che possa rendere il testo in formato numerico, interpretabile e utilizzabile dai modelli statistici e di *machine learning*. Per convertire il testo in una forma numerica che possa essere utilizzata come input per tali modelli sono state implementate diverse soluzioni. Nelle sezioni seguenti vengono presentati due approcci diversi finalizzati a questo scopo che verranno considerati come input nei modelli di classificazione.

3.1.1 Term Frequency - Inverse Document Frequency (*tf-idf*)

Il *tf-idf* è una tecnica molto comune utilizzata per valutare l'importanza di una parola in un documento all'interno di una raccolta di documenti. Questa metrica tiene conto sia della frequenza della parola nel documento (tf), che della frequenza inversa della parola in tutta la raccolta di documenti (idf). L'obiettivo è attribuire un punteggio più alto alle parole che sono importanti per un singolo documento, ma non troppo comuni nell'intera raccolta. In questo modo è possibile attribuire a ciascuna parola presente nel vocabolario un valore *tf-idf* specifico, per ciascun documento, e ottenere quindi una rappresentazione numerica. La variabile testuale può essere quindi strutturata in una matrice in cui per riga si hanno i documenti, per colonna le parole e in ogni cella il rispettivo valore *tf-idf* normalizzato. Dato che spesso alcuni termini sono presenti in pochi documenti, la matrice risultante sarà caratterizzata da un elevato grado di sparsità. Considerando un corpus D formato da N documenti aventi un vocabolario composto da T parole (o termini), l'indice *tf-idf* per il termine t_j (per $j = \{1, \ldots, T\}$) del documento d_i (per $i = \{1, \ldots, N\}$) viene definito come:

$$tf - idf(t_j, d_i, D) = tf(t_j, d_i) \times idf(t_j, D) \tag{3.1}$$

Dove:

- $tf(t_j, d_i)$ rappresenta la frequenza del termine t_j nel documento d_i e viene calcolato come:
$$tf(t_j, d_i) = \frac{f_{t_j,d_i}}{\sum_{t' \in d_i} f_{t',d_i}}$$
Dove f_{t_j,d_i} è la frequenza del termine t_j nel documento d_i e $\sum_{t' \in d_i} f_{t',d_i}$ è la somma delle frequenze di tutti i termini nel documento d_i.

- $idf(t_j, D)$ rappresenta l'importanza relativa del termine t_j nell'intero corpus e viene calcolato come:
$$idf(t_j, D) = \log \frac{N}{|\{d \in D : t_j \in d\}|}$$
In cui la quantità $|\{d \in D : t \in d\}|$ è il numero di documenti che contengono il termine t_j

Una volta calcolato ciascun valore $tf - idf$ per ogni termine, per ogni documento, si procede con la normalizzazione di cui la più comune è quella $L2$ che consiste nel rapporto tra il valore e la propria norma 2. Questo passaggio è necessario per garantire che i documenti con lunghezze diverse o con distribuzioni di frequenza dei termini diverse possano essere comparati in modo equo.

3.1.2 Sentence Embeddings

I *Large Lenguage Model*, di cui ne è stato descritto l'esempio di BERT nel capitolo precedente, forniscono un metodo alternativo per rappresentare i documenti del corpus in forma numerica preservando le relazioni semantiche che intercorrono tra le parole che li compongono: la rappresentazione *Sentence Embeddings* (o *Word Vectors*). L'idea alla base di tale trasformazione è proprio quella di sfruttare le capacità degli *LLM* preaddestrati per rappresentare i documenti in uno spazio vettoriale di dimensione fissa in cui la similarità tra vettori corrisponde alla similarità semantica tra documenti.

Consideriamo il corpus descritto in precedenza per il calcolo del $tf - idf$. Il processo per il calcolo del *sentence embedding* del generico documento d_i utilizzando un generico *LLM*, può essere articolato nei seguenti passaggi:

1. **Tokenizzazione**: d_i viene suddiviso in token secondo le regole di tokenizzazione dell'*LLM*.

2. **Aggiunta di token di supporto**: si aggiungono i token funzionali che aiutano il modello ad apprendere la struttura del documento (come ad esempio *[CLS]* e *[SEP]* per BERT).

3. ***Padding* e Troncamento**: si rimuovono gli ultimi token (troncamento) o si aggiungono token di supporto *[PAD]* in coda alla sequenza (*padding*) in modo tale da avere documenti composti dallo stesso numero di token. Tale dimensione dipende dallo specifico *LLM* e corrisponde alla dimensione dell'ultimo strato del modello. Ne consegue che sarà la stessa del *sentence embedding* finale.

4. **Applicazione del modello**: il testo tokenizzato della dimensione corretta viene dato in input al modello preaddestrato che la elabora fornendo in output la sua rappresentazione in forma di *sentence embedding*.

Anche allo scopo di calcolare *embeddings* efficaci viene spesso usata la tecnica del *fine-tuning* in modo tale da far apprendere al modello alcune specificità dei dati in esame.

Per dimostrare la capacità di queste rappresentazioni di evidenziare la similarità tra testi effettivamente semanticamente simili, viene presentato un semplice esempio applicativo. Consideriamo sei frasi distinte di cui due trattano di frutta, due parlano di cani, una di stelle e una di tecnologia. Lo scenario desiderato è quello per cui i vettori densi delle due coppie di frasi semanticamente simili siano vicini tra loro in termini di similarità e appaiano, invece, distanti le rappresentazioni delle frasi che trattano gli altri temi.

Per quantificare la similarità tra i vettori viene introdotta la Similarità Coseno: una tecnica euristica per la misurazione della similitudine tra due vettori. Questo indice è spesso utilizzato per valutare la similarità tra *embeddings* ed è basato sull'idea che vettori saranno tanto più simili quanto più sono allineati in termini di direzione nello spazio vettoriale. La similarità coseno tra i due vettori $\mathbf{A}$ e $\mathbf{B}$ è definita come:

$$\text{Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Una rappresentazione della similarità tra i sei embeddings considerati nell'esempio è visibile in Figura 3.1. Per realizzare il grafico sono stati calcolati gli *embeddings* delle sei frasi attraverso il modello *BERT-base* e da essi è stata definita la matrice delle similarità. A partire da essa è stata definita la matrice delle distanze i cui valori sono calcolati come complemento ad 1 delle similarità coseno. Ciascun vettore è stato poi proiettato e rappresentato su un piano bidimensionale attraverso la tecnica del *Multidimensional Scaling* (MDS), molto diffusa per la rappresentazione di vettori in spazi di dimensione ridotta.

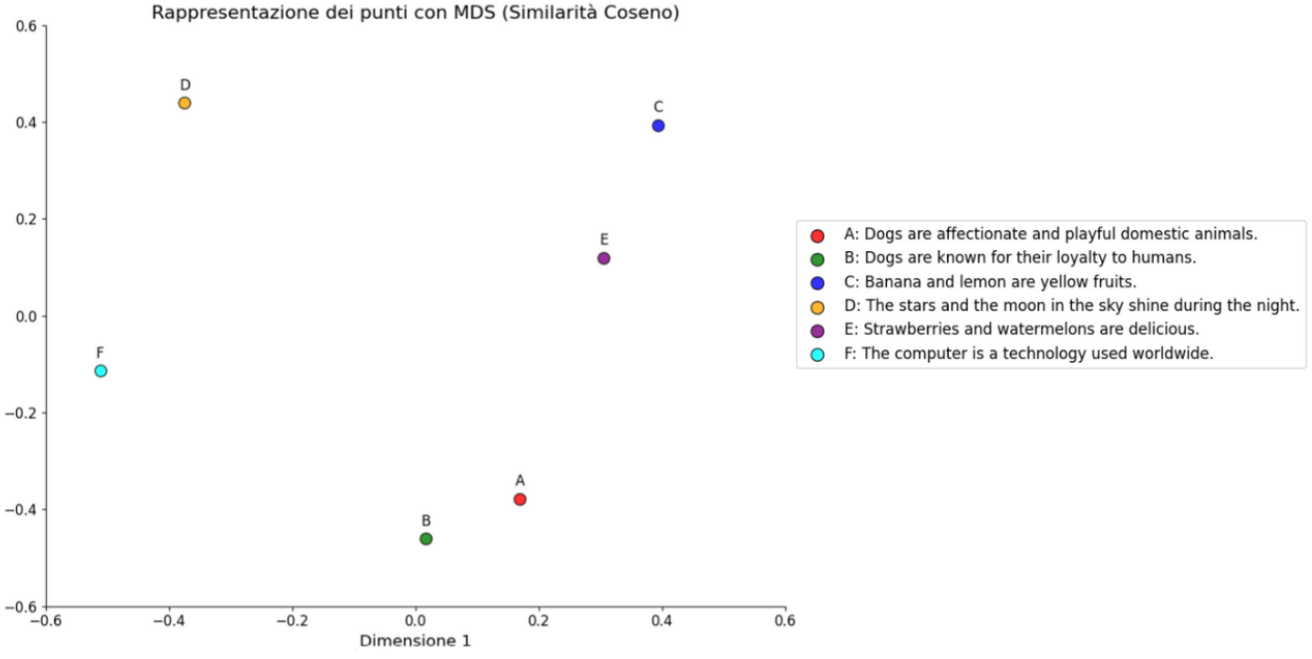

FIGURA 3.1: Disposizione dei punti attraverso MDS.

Dal grafico si vede come la similarità tra *embeddings* rispecchi la similarità semantica. Si notano infatti i punti delle due frasi sui cani (A e B), semanticamente molto simili, sono molto vicini tra loro e, al contrario, sono molto distanti dal punto D che si riferisce

alla frase sulle stelle. Allo stesso modo, le frasi C ed E, entrambe sul tema della frutta sono vicine nel piano e distanti da tutte le altre.

3.1.3 Rappresentazioni considerate nell'analisi

In questo studio di classificazione testuale viene posta molta attenzione al tipo di rappresentazione utilizzata. Ne vengono considerate diverse in modo tale da definire la matrice dei regressori che porta ad una classificazione migliore. Le alternative vagliate coinvolgono il *tf-idf*, rappresentazione attualmente implementata nei sistemi di classificazione automatica dei ticket negli applicativi Pat e diversi modelli preaddestrati basati su BERT che differiscono tra loro per la complessità misurata in termini di numero di parametri e conseguente costo computazionale, dimensione di output e lingua dei testi usati per il preaddestramento. In Tabella 3.1 vengono riassunte queste caratteristiche per i quattro modelli utilizzati.

I modelli in questione pur essendo tutte varianti dello stesso LLM presentano peculiarità che li contraddistinguono. Il **multi-qa-MiniLM-L6-cos-v1** (Hugging Face (2023b)) è uno dei modelli "BERT-*based*" più snelli e rappresenta l'alternativa più veloce in quanto, a parità di dataset, ha un tempo di calcolo degli *embeddings* pari ad un terzo rispetto a quello impiegato dal modello più complesso ed è interamente addestrato con testi in lingua inglese. Al contrario, **LaBSE** (Hugging Face (2023a)) è il modello più complesso che appunto necessita di elevata potenza computazionale. Esso però è molto usato per la sua capacità di adattarsi a diverse lingue in quanto per l'addestramento ne sono state considerate 109. Infine, **BERTino** (Matteo Muffo (2023)) è una versione di BERT addestrata su corpora italiani e detiene una complessità che lo pone tra i due modelli appena citati. Per BERTino è stata considerata anche la versione con *fine-tuning* (BERTino F.T.) in quanto si considera più efficace nel modificare i propri pesi sfruttando dati in italiano oltre ad essere un buon compromesso in termini di complessità.

Modello	**# Parametri**	$d_{\mathbf{model}}$	**Lingua**
multi-qa-MiniLM-L6-cos-v1	22,713,216	384	Inglese
BERTino	67,576,320	768	Italiano
LaBSE	470,926,848	768	Multilingua
BERTino F.T.	67,576,320	768	Italiano

Tabella 3.1: LLM considerati per il calcolo degli *embeddings*

3.2 Pulizia del testo

Un'ulteriore caratteristica dei dati testuali che rende complessa la loro elaborazione è la significativa presenza di rumore al loro interno. Ciascun testo, oltre alle parole, può contenere numerosi altri elementi che non portano valore in termini di significato specialmente se tale testo è una segnalazione espressa tramite un ticket scritto da un utente non soggetto a revisione. Oltre a queste componenti anche molte parole sono poco informative o di basso valore semantico definite "*stopwords*". Aggettivi e avverbi comuni, articoli, preposizioni, congiunzioni, verbi, pronomi e parole troppo o troppo poco frequenti, seppur fondamentali a livello strutturale, non arricchiscono la frase di significato e diventano elemento di disturbo nell'analisi testuale.

Non esiste una procedura standard di *preprocessing* nell'ambito NLP ma essa varia a seconda dell'applicazione. I passaggi più diffusi a tal scopo e che saranno considerati in questa analisi sono quelli schematizzati di seguito e applicati a titolo d'esempio nella Tabella 3.2:

- **Tokenizzazione**: si rimanda alla spiegazione fornita nella Sezione 2.3;
- **Conversione in *lowercase***: passaggio necessario per evitare che parole che differiscono solo dall'essere o avere caratteri in maiuscolo piuttosto che minuscolo vengano considerate parole diverse;
- **Rimozione di elementi di disturbo**: vengono identificati e rimossi tutti quei pattern di caratteri che non portano significato come email, numeri di telefono, collegamenti ipertestuali, tag HTML ed emoticon. In questa fase viene rimossa la punteggiatura e altri caratteri non alfanumerici che aumentano la complessità dei dati senza necessariamente contribuire al suo significato;
- **Rimozione delle *stopwords***: a partire da una lista di *stopwords* generalmente della lingua usata si estraggono le casistiche citate in precedenza di parole non informative. Molto spesso viene costruita una *stoplist* ad hoc inserendo parole che non arricchiscono la frase di significato in quel contesto d'analisi ma lo farebbero in altri;
- **Lemmatizzazione**: procedura in cui le parole vengono convertiti alla loro forma base detta "lemma". Prefissi, suffissi, coniugazioni verbali sono solo esempi di variazioni delle parole che portano grande variabilità all'interno dei dati senza aggiungere valore semantico; la lemmatizzazione opera proprio per limitare il più possibile questa componente di variabilità.

I passaggi appena descritti sono stati applicati anche ai dati considerati nell'analisi con lo scopo di ottenere una classificazione più accurata possibile. Essendo gli LLM allenati ad estrarre il significato semantico dei testi anche senza l'utilizzo di preprocessing, però, si è deciso di considerare come input per i modelli di classificazione anche *sentence embeddings* calcolati sui testi non preprocessati in modo tale valutare se l'integrità dei testi porti comunque significato e una classificazione ottimale.

Frase originale	"Ciao, sto riscontrando problemi con l'accesso al sistema. Ecco il mio indirizzo email: Mario.Rossi@mail.com"
Tokenizzazione	["Ciao", ",", "sto", "riscontrando", "problemi", "con", "l'accesso", "al", "sistema", ".", "Ecco", "il", "mio", "indirizzo", "email", ":", "Mario.Rossi@mail.com", "."]
Conversione in lower case	["ciao", ",", "sto", "riscontrando", "problemi", "con", "l'accesso", "al", "sistema", ".", "ecco", "il", "mio", "indirizzo", "email", ":", "mario.rossi@mail.com", "."]
Rimozione elementi di disturbo	["ciao", "sto", "riscontrando", "problemi", "con", "l'accesso", "al", "sistema", "ecco", "il", "mio", "indirizzo", "email"]
Rimozione delle stopwords	["riscontrando", "problemi", "l'accesso", "sistema", "mio", "indirizzo", "email"]
Lemmatizzazione	["riscontrare", "problema", "accesso", "sistema", "mio", "indirizzo", "email"]

Tabella 3.2: Esempio di applicazione del pre-processing ad una frase

Capitolo 4

Classificazione dei Ticket

Lo scenario applicativo su cui si basa questo studio, come introdotto nel Capitolo 1, coinvolge tre dataset distinti per l'*Incident detection* e due per la *Problem detection*. Il confronto delle prestazioni dei modelli, inoltre, avverrà separatamente per le due componenti del problema, in modo tale da individuare la miglior configurazione per il caso specifico. I modelli di classificazione considerati per la previsione della variabile risposta in entrambi i contesti di regressione binaria e multiclasse sono la Regressione Logistica, la *Support Vector Machine*, la Foresta Casuale (*Random Forest*) e la Rete Neurale ad unico strato latente. Per una descrizione teorica dei principi e delle metodologie di stima dei primi tre modelli menzionati si rimanda all'appendice A, per la Rete Neurale, invece, alla trattazione presente nel Capitolo 2.

4.1 Metriche di Valutazione

Prima di poter effettuare gli esperimenti che permettano di determinare quali siano le migliori codifiche dei dati testuali in esame e i migliori modelli di classificazione, è necessario definire le metriche che quantificano la capacità predittiva di ciascuna configurazione.

Nell'ambito dei problemi di classificazione, assume un ruolo centrale la *matrice di confusione* (o matrice di *errata classificazione*): uno strumento che permette di visualizzare facilmente le prestazioni dei modelli. Essa è una matrice di contingenza di dimensione, nel caso binario, 2×2 in cui vengono riportate le frequenze dei valori predetti correttamente e non correttamente per ciascuna classe. Considerando, come nel nostro caso, di voler distinguere n ticket tra "*Incident*" (classe positiva) e "Non *Incident*" (classe negativa) la matrice viene definita come

Predetti	**Osservati** Incident	Non Incident	Totale
Incident	Veri Positivi (TP)	Falsi Positivi (FP)	Inc. Predetti
Non Incident	Falsi Negativi (FN)	Veri Negativi (TN)	Non Inc. Predetti
Totale	Inc. Osservati	Non Inc. Osservati	n

Spesso è utile tenere in considerazione la versione probabilistica della matrice di confusione calcolata dividendo ciascun termine per il totale di colonna. In questo caso si pone: $TP = 1 - \beta$, $FN = \beta$, $TN = 1 - \alpha$ e $FP = \alpha$.
A partire da questa tabella vengono calcolate le metriche di valutazione più diffuse in questo ambito e che verranno considerate in questo studio:

- **Accuratezza**: $\frac{TP+TN}{n}$
- **Sensibilità (o *Recall*)**: $\frac{TP}{TP+FN} = 1 - \beta$
- **Specificità**: $\frac{TN}{TN+FP} = 1 - \alpha$
- **Precisione**: $\frac{TP}{TP+FP} = \frac{1-\beta}{1-\beta+\alpha}$
- **F1-score**: $\frac{2TP}{2TP+FP+FN} = \frac{2}{\frac{1}{\text{Recall}}+\frac{1}{\text{precisione}}}$

L'accuratezza è la misura più immediata e quantifica la proporzione di unità del campione di riferimento predette correttamente sul totale, è particolarmente indicata nei casi in cui le classi sono bilanciate. Sensibilità e specificità, invece, indicano la proporzione di unità positive e negative rispettivamente predette correttamente sul totale delle unità osservate per quella classe; risultano molto adatte nei contesti in cui una corretta previsione di una classe ha maggiore importanza di una corretta previsione sull'altra. La precisione è calcolata come la proporzione di unità predette correttamente per la classe positiva sul totale delle previsioni positive. Il suo utilizzo è consigliato nei casi in cui si vuole minimizzare il numero di falsi positivi. Infine, l'F1-score è calcolato attraverso la media armonica tra Precisione e Recall e, proprio per questo motivo, è particolarmente indicato in situazioni in cui si cerca un equilibrio tra le sue due componenti come quando le conseguenze di falsi positivi e falsi negativi sono simili. Come funzione di aggregazione viene scelta la media armonica perché penalizza fortemente i valori estremi, garantendo che un F1-score sufficientemente elevato sia possibile solo se sia Precisione che Sensibilità sono alti. Questo aspetto permette alla metrica di essere maggiormente informativa quando si considerano dataset con classi sbilanciate. La situazione considerata in questo studio è proprio quella che vede l'impiego dell'F1-score come la scelta ottimale per il

criterio di valutazione in quanto l'impatto sul sistema IM della classificazione errata di un *Incident* o di una *Service Request* è il medesimo e, come mostrato al paragrafo 1.3.1, i dataset sono fortemente sbilanciati.

È doveroso puntualizzare che le metriche sopra descritte vengono intese a parità di soglia decisionale, dove quest'ultima è il valore con cui vengono confrontati gli output dei modelli, oltre il quale la singola osservazione viene predetta di classe positiva. Ne consegue che la scelta della soglia decisionale è un aspetto cruciale in un problema di classificazione binaria, poiché influenza direttamente le metriche di valutazione. Molto spesso la soglia viene posta a $0,5$ o pari alla proporzione campionaria della classe positiva. Allo scopo di mitigare ulteriormente gli effetti dati dallo sbilanciamento delle classi della variabile risposta sui risultati, in questo studio la scelta della soglia viene effettuata sulla base della Curva ROC.

La curva ROC (*Receiver Operating Characteristic*) è uno strumento grafico di frequente utilizzo per valutare le prestazioni di un classificatore binario. Essa rappresenta la relazione tra la Sensibilità e il complemento a 1 della specificità a diversi livelli di soglia decisionale. Ogni punto sulla curva ROC corrisponde a una coppia $(1 - \text{Specificità}, \text{Sensibilità})$ per una determinata soglia decisionale. L'area sottesa sotto questa curva (AUC) è un valore spesso usato per valutazioni globali sui modelli di classificazione indipendentemente dalla soglia decisionale.

In questo lavoro la scelta avviene selezionando il valore che minimizza la distanza dalla curva ROC al punto $(0, 1)$ (Figura 4.1). Questa procedura mira a bilanciare i tassi di falsi positivi e veri positivi; il punto $(0, 1)$ rappresenta, infatti, la situazione ideale in cui si registrano Sensibilità e Specificità massime. L'adozione dell'F1-score come metrica di valutazione congiuntamente alla procedura di selezione ottimale della soglia permette di assumere che l'impatto dello sbilanciamento delle classi venga adeguatamente gestito, evitando tecniche di ricampionamento che alterano la distribuzione originale dei dati, preservando così le caratteristiche intrinseche del dataset e riducendo il rischio di introdurre dipendenze deterministiche tra le osservazioni.

Nell'ambito del *Problem Detection* non è possibile considerare direttamente le metriche descritte nel paragrafo precedente in quanto utilizzabili solamente nei problemi classificazione binaria. In questo caso l'indicatore delle prestazioni dei modelli che viene utilizzato è l'estensione al caso multiclasse del F1-score denominata F1-score pesato (*weighted*). Considerando insieme di dati di numerosità campionaria n, in un problema di classificazione a K classi, in cui $n_1, \ldots, n_K$ sono le frequenze osservate di ciascuna

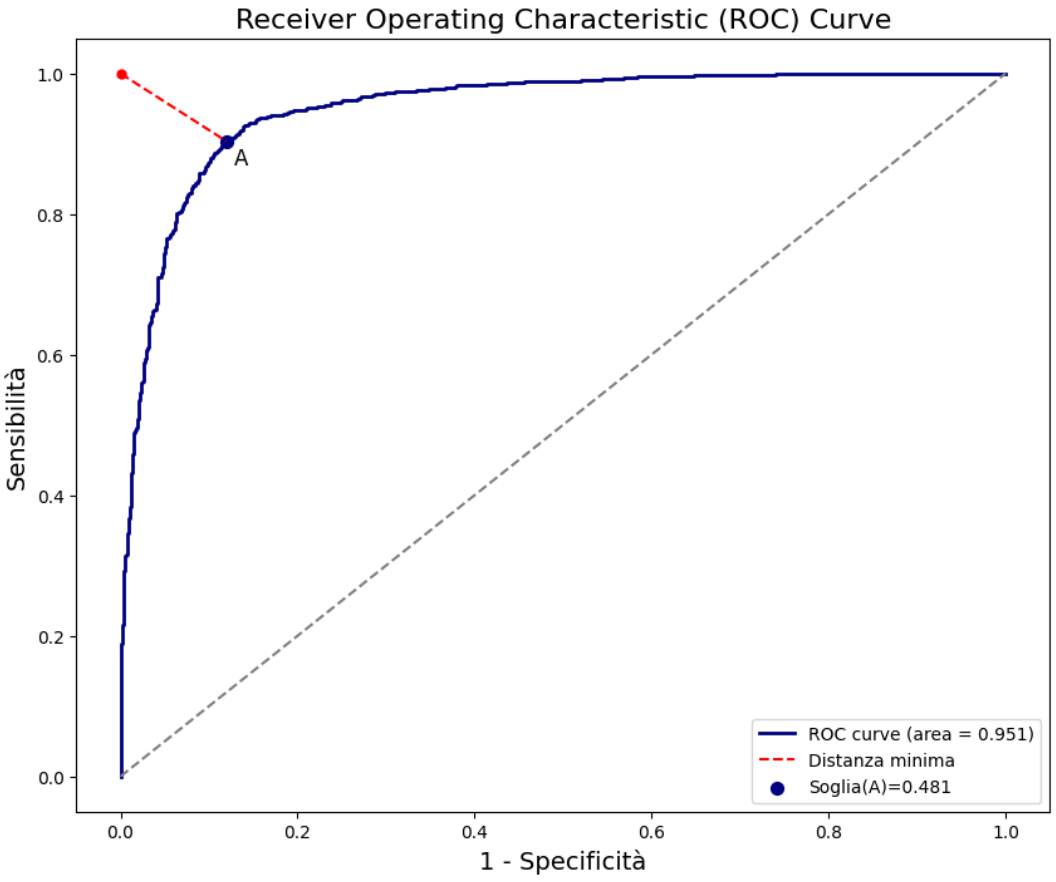

FIGURA 4.1: Curva ROC con evidenza della soglia ottimale

classe nel campione, l'F1-score pesato viene calcolato come

$$\mathrm{F1}_{weighted} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{F1}_k$$

in cui $\mathrm{F1}_k$ è l'F1-score calcolato considerando la classe k come classe positiva e le rimanenti classe negativa. Le ragioni che giustificano la scelta di questa metrica anche in questo caso sono le medesime del precedente. La frequenza delle classi all'interno dei due dataset considerati, infatti, sono fortemente sbilanciate. Basti considerare che la frequenza minima e la frequenza massima osservate per il dataset `Farmaceutico` sono 697 e 41, mentre per il dataset `Multiservizi` sono 639 e 35.

4.2 Incident Detection

Come già anticipato nei capitoli precedenti, gli esperimenti condotti in questo studio considerano, per ciascun insieme di dati, tutte le combinazioni di rappresentazioni

numeriche dei testi e modelli di classificazione. Gli *embedder* basati su LLM che non subiscono la procedura di *fine-tuning* sono comuni ai tre dataset invece, per costruzione, l'*embedder* BERTino *fine-tuned* è specifico. Per la fase di ottimizzazione dei pesi di BERTino è stato utilizzato un approccio supervisionato in cui la rete è stata addestrata sull'insieme di stima e validata attraverso l'insieme di convalida per un numero ottimale di epoche, dato un *learning rate* fissato pari a 2×10^{-6} minimizzando la *loss-function* BCE. L'addestramento di modelli ad un così elevato numero di parametri necessita di risorse computazionali significative. Per limitare l'impatto di questo fattore sullo studio è stata sfruttata la naturale parallelizzazione della rete sfruttando l'architettura GPU che ha abbattuto drasticamente il tempo di esecuzione. Nella Figura 4.2 vengono riportati i grafici in cui si mostra l'andamento della funzione di perdita sull'insieme di stima e sull'insieme di convalida al variare di 10 epoche di addestramento evidenziandone il numero ottimale che è stato poi utilizzato per la stima dell'*embedder* finale.

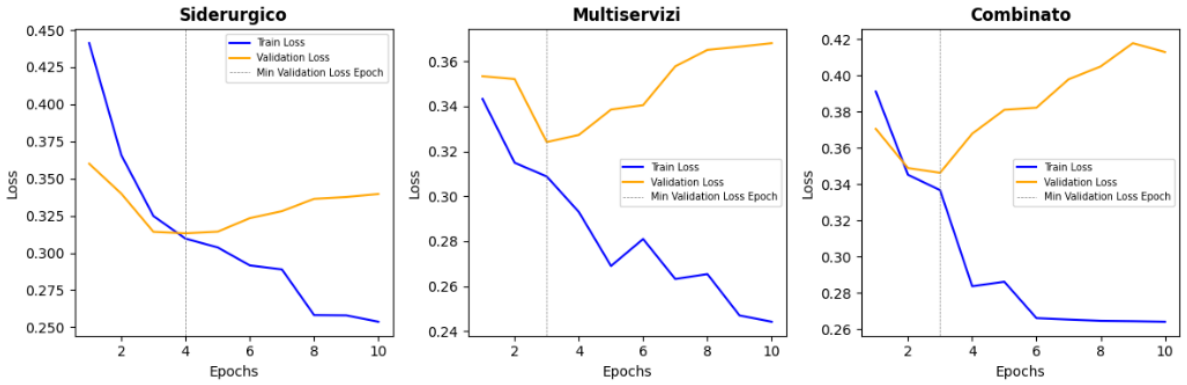

FIGURA 4.2: Train Loss ed Evaluation Loss fine-tuning per epoca.

Ad ogni *corpus*, è stata applicata la procedura di *preprocessing* descritta al paragrafo 3.2. In questo modo si considerano 9 rappresentazioni differenti su cui stimare i quattro modelli di classificazione: per ogni *embedder* una versione sui dati grezzi, ed una su quelli processati. La rappresentazione mancante tra le dieci possibili è il BERTino *fine-tuned* di cui viene calcolata solamente la prima versione. Ciò viene effettuato per valutare l'effetto della pulizia dei dati sulla capacità di creare proiezioni informative, aspettandosi un miglioramento per il *tf-idf* dopo aver fatto *preprocessing* e nessun miglioramento o un possibile peggioramento per gli LLM pre-addestrati.

Per trovare il miglior modello per ciascuna classe di modelli considerati è necessario individuare la configurazione di iperparametri che garantisce capacità predittive ottimali. In generale questo tipo di procedura si declina nello stimare un elevato numero

di modelli diventando così estremamente onerosa a livello computazionale. Per questa ragione come prima valutazione quantitativa delle performance di tutte le possibile combinazioni sono stati stimati tutti i classificatori con una parametrizzazione di default. Per le Regressioni Logistiche è stata considerata la penalizzazione Ridge avente $\lambda_{L2} = 1$; per le Foreste Casuali un numero di variabili da ispezionare in ogni nodo pari a $F = \sqrt[2]{p}$ per 100 alberi. La funzione kernel adottata dalle Support Vector Machines (SVM) è quella a base radiale (RBF) con parametro di scala $\lambda = 1/\left(p \cdot \sigma^2_{X_{Conc}}\right)$, dove X_{Conc} è la concatenazione delle colonne che formano la matrice X, e costo di errata classificazione $\gamma = 1$. Infine, lo strato nascosto della rete neurale è formato da 64 nodi e considera come funzioni di attivazione $f_o(x) = ReLU(x)$ ed $f_1(x) = Sigmoide(x)$; per il suo addestramento è stato considerato un numero di epoche pari a 30 senza dropout e penalizzazione, con un tasso di apprendimento $\eta = 0.001$ e dimensione di mini-batch fissata a 16. In Appendice B vengono riportate le tabelle contenenti nel dettaglio i valori di F1-score, Recall, soglia decisionale ottimizzata e tempo di stima e valutazione del modello per ciascuna configurazione considerata per ciascun dataset.

Dai risultati ottenuti dall'applicazione dei modelli sui dataset `Siderurgico`, `Multiservizi`, e `Combinato`, emerge che il *preprocessing* ha un effetto positivo sui modelli addestrati con *tf-idf*, migliorando significativamente le loro prestazioni rispetto a quelli addestrati su dati grezzi. Al contrario, per gli input basati su LLM, i *sentence-embeddings* calcolati sui dati grezzi producono indici predittivi migliori. La soglia decisionale ottimale varia con la configurazione, ma tendenzialmente si avvicina alla proporzione campionaria degli *Incident*, indicando l'importanza dell'ottimizzazione specifica per ogni configurazione.

Per quanto riguarda il tempo d'esecuzione, esso dipende dal modello di classificazione adottato: l'SVM è il più oneroso, mentre la Regressione Logistica è la più leggera. Gli input basati su *tf-idf* richiedono tempi più lunghi a causa del maggior numero di colonne nella matrice dei regressori X. In termini di bontà predittiva, BERTino F.T. supera gli altri *embeddings*, con risultati paragonabili solo ai modelli con *tf-idf*. Pertanto, nelle fasi successive dello studio, si considereranno solo BERTino F.T. e *tf-idf* sui dati preprocessati.

Analizzando le differenze tra i dataset, per `Siderurgico` e `Multiservizi`, le prestazioni dei modelli seguono un trend simile, sebbene nel dataset `Multiservizi` i modelli stimati sull'embedder ottimizzato non si distinguano così marcatamente rispetto agli altri. Complessivamente, le prestazioni sul dataset `Siderurgico` risultano superiori. Nel caso del dataset `Combinato`, i risultati rispecchiano quelli dei singoli dataset componenti, con BERTino F.T. che mostra le migliori prestazioni, seguito dal *tf-idf*. Anche in questo caso, la *model selection* si concentrerà su queste due rappresentazioni numeriche

dei testi. L'SVM si conferma il classificatore con i migliori risultati per tutti gli input.

4.2.1 Grid Search e Risultati Finali

La fase di regolazione dei modelli (o *model-selection*), come già anticipato, consiste nella procedura di ricerca della configurazione degli iperparametri per ciascuna classe di modelli che garantisca prestazioni ottimali. I due approcci più popolari sono denominati *Grid Search* e *Random Search* ed entrambi si fondano sulla medesima idea di base: valutare i modelli stimandoli considerando le combinazioni di iperparametri rappresentati da una griglia di valori possibili. L'aspetto che li differenzia è il modo in cui vengono scelte le configurazioni: nel primo caso vengono ispezionate tutte, nel secondo caso la selezione è casuale. Ne consegue che se il primo è più esaustivo e computazionalmente oneroso, il secondo preclude combinazioni che possono potenzialmente essere ottimali ma è molto più snello.

La procedura attuata in questo studio si basa sulla *Grid Search*. Nello specifico per le rappresentazioni *tf-idf* e BERTino F.T., per ogni modello di classificazione, viene considerata una griglia iniziale di iperparametri comune a tutti i dataset di cui verranno valutate tutte le combinazioni. Viene quindi stimato ciascun modello e poi valutato tramite convalida incrociata a 4 *fold* su ciascun insieme di stima. Successivamente, a fronte dei risultati ottenuti per ogni caso specifico, la ricerca avverrà in maniera più fine considerando iterativamente una griglia di valori a partire dalla configurazione ottimale della prima griglia. I valori per ciascun parametro di ciascun modello che vanno a comporre la prima griglia vengono riportati nella Tabella 4.1 per dare un'idea della loro scala. I valori delle griglie successivi vengono omessi.

Modello di classificazione	**Iperparametro**	**Valori considerati**
Regressione Logistica	Penalizzazione	Ridge, Lasso, Elastic Net
	λ	0.1, 1, 10, 50, 100
Support Vector Machine	γ	0.1, 1, 10, 50, 100
	Kernel	Lineare, Base Radiale
	λ	0.1, 1, 10, 50, 100
Random Forest	F	$\sqrt[2]{p}$, 0.1, 1, 10, 50
	n. di alberi	50, 100, 200, 300
Rete Neurale	n. nodi	32, 64, 128, 256
	n. epoche	10, 30, 50, 100
	mini-batch size	4, 8, 16, 32, 128
	η	0.01, 0.001, 0.0001

Tabella 4.1: Valori esplorati che compongono la prima griglia

I modelli selezionati al termine della procedura *Grid Search* con relativi F1-score sono riassunti, per ciascun dataset, nella Tabella 4.2 riportata a fine sezione. Osservando i risultati finali relativi al dataset `Siderurgico`, si riscontra che il modello che ha ottenuto le migliori prestazioni complessive è stata la Rete Neurale addestrata su *embeddings* generati da BERTino F.T., con un F1-score di 0.898. Questo risultato conferma la grande efficacia della rappresentazione calcolata tramite BERTino nel catturare il significato semantico dei testi per la classificazione degli *Incident*. In generale, tutti i modelli addestrati con *embeddings* BERTino superano quelli addestrati con *tf-idf*, come avveniva anche prima della messa in atto della regolarizzazione dei modelli di classificazione. Va comunque posta attenzione sul fatto che, nel contesto del dataset in questione, la *Support Vector Machine* calcolata sul tf-idf sui dati preprocessati porta ad un indice F1 elevato e comparabile a quello dei modelli che considerano BERTino come input. In situazioni in cui si hanno tempo o capacità di calcolo limitati, calcolare tramite LLM le rappresentazioni vettoriali dei testi può risultare complicato trovando quindi nel *tf-idf* una valida alternativa veloce e più leggera a livello computazionale.

Spostando l'attenzione sui risultati per il dataset `Multiservizi`, la configurazione che riporta il miglior F1-score (0.852) è l'SVM addestrato su *embeddings* BERTino. Concentrandosi sugli altri tre modelli di classificazione, la differenza delle performance date dai due diversi input è meno marcata rispetto a quanto accadeva per il dataset `Siderurgico`. Per il Random Forest, ad esempio, è preferibile la rappresentazione *tf-idf*. Anche in questo caso si può concludere che *sentence embeddings* calcolati attraverso LLM a cui è stata applicata la procedura di *fine tuning* portano a risultati migliori rispetto alle rappresentazioni più classiche, anche se non in maniera così marcata da ritenere inverosimili situazioni in cui sono comunque da preferire quest'ultime per privilegiare la velocità calcolo.

Infine, nel dataset `Combinato`, gli *embeddings* BERTino hanno ancora una volta prodotto i migliori risultati. La SVM e la Rete Neurale hanno entrambi raggiunto l'F1-score più elevato, pari 0.911. In questo caso è molto marcata la differenza tra i due diversi input e si arriva ad indici più elevati rispetto ai dataset precedenti. Una spiegazione è sicuramente data dal fatto che, considerando l'unione di due dataset la quantità di dati utilizzati per la stima dei modelli è maggiore. È interessante notare che i risultati dei modelli che considerano BERTino F.T. sono molto vicini tra loro, indicando quanto abbia più peso sull'accuratezza della classificazione un input informativo rispetto alla scelta del classificatore finale. In uno scenario aziendale in cui si vuole costruire un sistema di *Incident Detection* automatica, un ruolo centrale risiede, quindi, nell'allestire un LLM ottimizzato considerando dati da diversi clienti in modo tale da avere un

embedder specializzato sul dominio di ricerca (l'*IT Service Management*) ma che non catturi eccessivamente le specificità semantiche dei singoli clienti.

Volendo trarre delle conclusioni più generali dai risultati ottenuti, che vadano oltre al condizionamento al singolo dataset, gli *embeddings* generati da BERTino F.T. si sono dimostrati superiori rispetto ai *tf-idf* in tutti i dataset. Seppur meno impattante della scelta dell'input, la scelta del classificatore migliore ricade sull'SVM che porta alle prestazioni più elevate. Si sottolinea, infine, che il processo di ottimizzazione degli iperparametri condotto ha portato a benefici in termini di F1-score sottolineando l'importanza della loro regolazione anche se, per rendere questa fase più completa, sarebbe stato necessario approfondirla considerando, per ciascun modello, per ciascun input, per ciascun dataset, un set di possibili configurazioni più esaustivo.

Dataset	**Input**	**Modello**	**F1**
`Siderurgico`	tf-idf + prep.	Regressione Logistica	0.879
		Support Vector Machine	**0.889**
		Random Forest	0.859
		Rete Neurale	0.843
	BERTino F.T.	Regressione Logistica	0.891
		Support Vector Machine	0.894
		Random Forest	0.897
		Rete Neurale	**0.898**
`Multiservizi`	tf-idf + prep.	Regressione Logistica	0.814
		Support Vector Machine	**0.823**
		Random Forest	0.817
		Rete Neurale	0.802
	BERTino F.T.	Regressione Logistica	0.819
		Support Vector Machine	**0.852**
		Random Forest	0.800
		Rete Neurale	0.805
`Combinato`	tf-idf + prep.	Regressione Logistica	0.848
		Support Vector Machine	**0.873**
		Random Forest	0.845
		Rete Neurale	0.857
	BERTino F.T.	Regressione Logistica	0.909
		Support Vector Machine	**0.911**
		Random Forest	0.902
		Rete Neurale	0.911

Tabella 4.2: F1-score modelli selezionati per i diversi dataset

4.3 Problem Detection

La struttura dell'analisi condotta per la valutazione dei modelli e delle rappresentazioni numeriche dei testi per il contesto della *Problem Detection* è la medesima adottata per l'*Incident Detection*. Analogamente a quanto accaduto per classificazione binaria, per il fine tuning di BERTino è stato utilizzato un approccio supervisionato utilizzando l'insieme di stima per addestrare il modello e l'insieme di convaida per valutarlo a ogni epoca. La funzione di perdita che viene minimizzata considerando un *learning rate* valorizzato a 2×10^{-6} è la CE. Impiegando l'architettura GPU per parallelizzare l'onere computazionale del fine-tuning, la rete è stata addestrata per 10 epoche sia per il dataset `Farmaceutico` sia per il dataset `Multiservizi` in modo tale da ricavarne il numero ottimale. L'andamento della funzione di perdita nell'insieme di stima e di convalida per i due dataset è rappresentato in Figura 4.3 da cui si deriva che il numero di epoche di addestramento da preferire ed effettivamente considerato per i due dataset è 7 nel caso `Farmaceutico` e 4 nel caso `Multiservizi`.

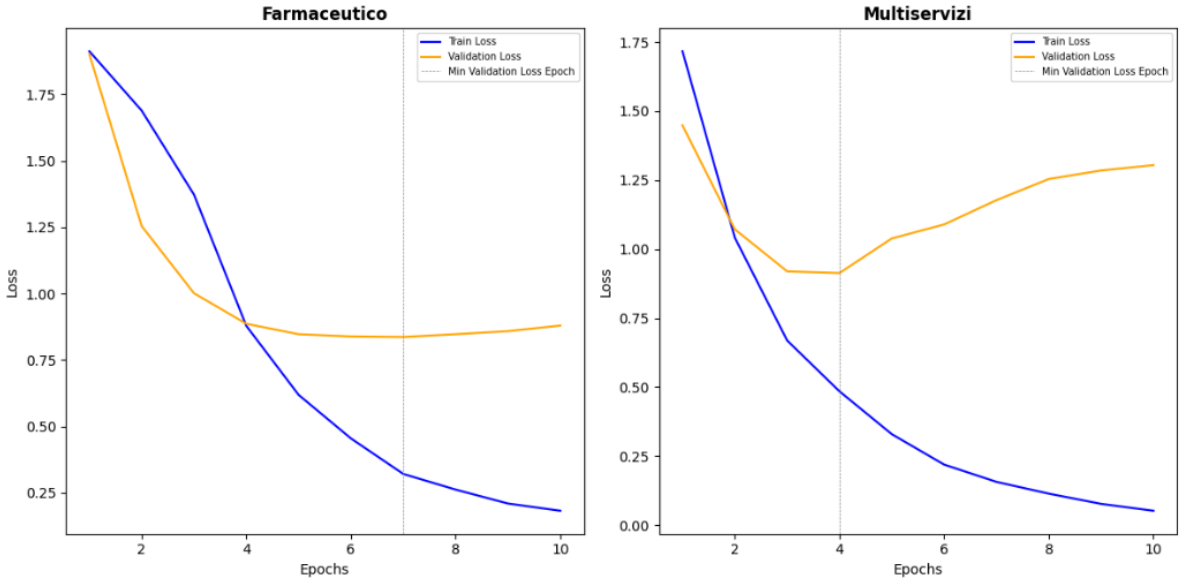

FIGURA 4.3: Train Loss ed Evaluation Loss fine-tuning per epoca.

Le configurazioni considerate coinvolgono le stesse 9 rappresentazioni alternative del caso binario e gli stessi modelli di classificazione. I classificatori sono stati stimati per ciascun input in prima battuta utilizzando come valori iniziali degli iperparametri quelli elencati al paragrafo 5.2. L'unica differenza risiede nella funzione di attivazione f_1 della rete neurale che, data la natura del problema, è stata definita $f_1(x) = Softmax(x)$. L'idea è sempre quella di quantificare la capacità predittiva di ciascuna configurazione

in modo tale da focalizzarsi successivamente solo sull'ottimizzazione degli iperparametri che regolano i modelli di classificazione di quelle configurazioni che portano ai risultati migliori. Sempre in Appendice B si possono trovare nel dettaglio i valori di F1-score pesato, Accuracy e tempo di stima e valutazione del modello per ciascuna configurazione considerata per ciascun dataset.

I modelli adattati ai diversi input per il dataset `Farmaceutico` portano a buoni livelli di F1-score ed accuratezza considerando il fatto che la variabile d'interesse è composta da 50 classi distinte. Se venisse effettuata una classificazione casuale in cui ogni classe ha la medesima probabilità di essere selezionata, infatti, l'accuratezza attesa sarebbe pari a 0.02. Confrontando i risultati ottenuti per i diversi input, al netto dei differenti modelli di classificazione, si riscontra come anche in questo caso gli *embeddings* calcolati tramite LLM che non hanno subito *fine-tuning* portano a prestazioni nettamente inferiori rispetto a quelle ottenute impiegando BERTino F.T. o la rappresentazione *tf-idf*. Si evidenzia così anche nel contesto della classificazione multiclasse il fatto che, per la tipologia dei testi considerati in questo studio, LLM preaddestrati senza passaggi di ottimizzazione sui dati in esame siano troppo generali e fatichino ad estrapolare in maniera significativa il significato semantico insito nei testi. Come avveniva anche per l'*Incident Detection* l'effetto del preprocessing applicato ai testi prima della loro conversione in vettori numerici è positivo per il *tif-idf* e negativo per gli altri *embedders*. A livello di singola tipologia di input, le più competitive sono ancora una volta *tf-idf* sui dati preprocessati e BERTino F.T.. Osservando i modelli di classificazione, invece si nota un'ottima prestazione della Regressione Logistica nel caso del *tf-idf* che porta all'indice F1 più alto (0.745). Focalizzandosi invece sui valori di F1 riportati dai modelli adattati su BERTino F.T., l'SVM riporta quello più alto (0.776).

I risultati ottenuti sul dataset `Multiservizi` portano a considerazioni molto simili a quelle fatte per il dataset precedente. Si nota ancora infatti quanto appaiano inadeguati rispetto a BERTino F.T. e *tf-idf* i *sentence embeddings* calcolati dai modelli non ottimizzati portando alla scelta di quest'ultimi per la fase di regolazione dei modelli. Un aspetto curioso, però, è il fatto che sia per il LaBSE che per il multi-qa-MiniLM-L6-cos-v1, la versione calcolata sui dati preprocessati supera quella calcolata sui dati grezzi, seppur non di molto. Considerando marginalmente i modelli di classificazione si nota una differenza rispetto alle valutazioni fatte nel contesto dell'*Incident Detection*. Questa differenza consiste nell'aumento delle performance della Regressione Logistica. Se per la classificazione binaria il modello di classificazione da preferire in generale era l'SVM, ora la scelta ricade sulla Regressione Logistica. Tale aspetto è ancora più marcato per questo dataset rispetto al precedente. È doveroso puntualizzare, comunque, che questa

considerazione viene condotta prima di visionare i risultati ottenuti tramite i modelli con iperparametri selezionati in maniera ottimale. Questa prima analisi quantitativa evidenzia ancora una volta l'importanza dell'ottimizzazione dei LLM per il calcolo degli *embeddings* rispetto al contesto specifico. Anche per il dataset `Multiservizi` per la *Problem Detection*, il fine-tuning di modelli preaddestrati come BERTino F.T. si dimostra la strategia più efficace, mentre *tf-idf* con preprocessing rappresenta una valida alternativa per configurazioni meno complesse e computazionalmente onerose.

4.3.1 Grid Search e Risultati Finali

La fase di *model selection* attuata per la parte di classificazione multiclasse dello studio segue la medesima procedura delineata per la parte della classificazione binaria. Anche in questo caso, si ricorre alla metodologia *Grid Search* per identificare la configurazione ottimale degli iperparametri. Come nel caso precedente essa avviene esplorando una griglia iniziale di valori comune a tutti i dataset, stimando ciascun modello e valutandolo tramite convalida incrociata a 4 fold su ogni insieme di stima. Successivamente, i risultati ottenuti guideranno una ricerca più dettagliata, perfezionando iterativamente la griglia di valori a partire dalla configurazione ottimale iniziale. I valori per ciascun parametro di ciascun modello che compongono la prima griglia sono gli stessi utilizzati per la classificazione binaria riportati nella Tabella 4.1. Anche in questo caso, i valori considerati nelle griglie successive, create durante la fase iterativa, non sono riportati nei dettagli, ma dei modelli selezionati al termine della procedura di *Grid Search* vengono riportati relativi F1-score pesati, per ciascun dataset, nella Tabella 4.3 a fine sezione.

Osservando la tabella dei risultati relativi ai dataset `Farmaceutico` e `Multiservizi`, emerge chiaramente come l'input basato su BERTino F.T. superi in maniera consistente il *tf-idf* sui testi preprocessati in termini di F1-score pesato per entrambi i dataset. Osservando la la parte specifica del dataset `Farmaceutico`, infatti, si vede che il modello SVM addestrato su BERTino F.T. ottiene il miglior F1-score (0.788), seguito dalla Regressione Logistica (0.777). La migliore performance tra i modelli con input *tf-idf* viene ottenuta dalla Regressione Logistica con un F1-score di 0.752, dimostrando che, seppure efficace, non raggiunge i livelli dei modelli basati su BERTino F.T..

Per il dataset `Multiservizi` (Tabella 5.11), il divario tra le prestazioni gli input è ancora più marcato. La Regressione Logistica con BERTino F.T. ottiene un F1-score di 0.842, risultando il modello più performante. Anche qui, tutti i modelli basati su BERTino F.T. superano quelli basati su *tf-idf*, con il miglior F1-score tra questi ultimi ottenuto dal Random Forest (0.736).

Focalizzandosi sui modelli di classificazione, si osserva che nel dataset `Farmaceutico`

l'SVM è il modello che porta alle migliori prestazioni, mentre nel dataset `Multiservizi` la Regressione Logistica con BERTino F.T. emerge come la configurazione da preferire. Questo suggerisce che, se a livello di input la scelta ricade su BERTino F.T. a prescindere dal dataset, la scelta a livello di miglior classificatore può variare in funzione del dataset specifico.

Volendo trarre delle considerazioni più generali, si conferma che, anche nel caso di problemi di classificazione a più classi dei testi dei ticket come la *Problem Detection* in esame, gli *embeddings* generati da BERTino a cui è stato applicato *fine-tuning* sono superiori rispetto ai *tf-idf* in tutti i dataset esaminati. Questa superiorità è evidente anche nelle performance dei vari modelli di classificazione, con l'SVM che frequentemente raggiunge le prestazioni più elevate quando associato a BERTino F.T. Tuttavia, la scelta del modello ottimale può variare a seconda del contesto specifico del dataset. La fase di ottimizzazione degli iperparametri ha evidenziato la sua importanza, dimostrando nuovamente come una regolazione accurata possa incrementare significativamente le performance dei modelli.

Dataset	**Input**	**Modello**	**F1**
`Farmaceutico`	tf-idf + prep.	Regressione Logistica	**0.752**
		Support Vector Machine	0.735
		Random Forest	0.736
		Rete Neurale	0.747
	BERTino F.T.	Regressione Logistica	0.777
		Support Vector Machine	**0.788**
		Random Forest	0.769
		Rete Neurale	0.769
`Multiservizi`	tf-idf + prep.	Regressione Logistica	0.728
		Support Vector Machine	0.683
		Random Forest	**0.736**
		Rete Neurale	0.723
	BERTino F.T.	Regressione Logistica	**0.842**
		Support Vector Machine	0.821
		Random Forest	0.792
		Rete Neurale	0.823

Tabella 4.3: F1-score modelli selezionati per i diversi dataset

Conclusione

Il contesto in cui questa ricerca si inserisce è quello dell'*IT Service Management*, focalizzandosi sui processi di *Incident Detection* (classificazione binaria) e *Problem Detection* (classificazione multiclasse) applicato al caso aziendale di Pat SRL. Il problema affrontato è quello di migliorare l'accuratezza e l'efficienza dei sistemi di classificazione automatica integrati nel prodotto erogato da Pat HDA attraverso l'ispezione di tecniche avanzate di rappresentazione dei testi e l'ottimizzazione dei modelli di classificazione applicati ad esse.

Dopo una descrizione introduttiva dei sviluppi recenti delle tecniche di *Natural Language Processing* (NLP), è stata posta l'attenzione sul modello Rete Neurale introducendolo a partire dalla sua formulazione più semplice fino ad evoluzioni più complesse come l'architettura *Transformers*, trattata nel dettaglio nel contesto del *Large Language Model* BERT. È stata esplorata, poi, l'evoluzione delle rappresentazioni testuali dai metodi tradizionali basate sulla frequenza dei termini in un *corpus* come il tf-idf fino ai *sentence embeddings* generati dai modelli di linguaggio pre-addestrati, come quelli derivati da BERT considerati in questo studio. Dopo aver sottolineato l'importanza del pre-processing dei dati testuali per migliorare la qualità delle rappresentazioni, l'attenzione si è spostata sui diversi *embeddings* considerati e sulla descrizione della procedura di fine-tuning dei modelli di linguaggio per il calcolo di rappresentazioni ottimizzate per i dati considerati.

L'applicazione di queste tecniche ai dataset utilizzati ha portato a conclusioni significative. Gli *embeddings* calcolati attraverso il modello BERTino ottimizzato tramite *fine-tuning*, si sono dimostrati capaci di catturare in modo più efficace le informazioni semantiche presenti nei testi. Essi, infatti, hanno portato i modelli di classificazione a superare in modo consistente le performance dei modelli stimati su rappresentazioni più tradizionali come il *tf-idf*. Questo risultato sottolinea l'importanza di utilizzare, sia nel contesto binario che in quello multiclasse, rappresentazioni testuali avanzate che catturino al meglio le sfumature semantiche dei testi. Si puntualizza però il fatto che l'impiego diretto di modelli preaddestrati senza l'applicazione della procedura di *fine-tuning* porta

a prestazioni non paragonabili a quelle date da *tf-idf* o da modello ottimizzato in contesti specifici come quello considerato dimostrando quanto l'ottimizzazione dei parametri dell'*embedder* sia un passaggio preliminare fondamentale.
Considerando marginalmente l'effetto dei diversi classificatori, la fase di ottimizzazione degli iperparametri tramite *Grid Search* sequenziale si è rivelata cruciale per migliorare le prestazioni dei modelli di classificazione, evidenziando la propria importanza. Nonostante ciò, non è stata individuata una classe di modelli sempre preferibile alle altre a prescindere dal dataset di applicazione, mostrando come la scelta debba essere specifica del caso.

In generale, i risultati ottenuti suggeriscono che un approccio integrato, che combina rappresentazioni semantiche avanzate calcolate per mezzo di *Large Language Model* ottimizzati e modelli di classificazione opportunamente regolarizzati, è fondamentale per migliorare l'efficacia dei sistemi di *Incident* e *Problem Detection* nei contesti aziendali in cui è disponibile una sufficiente capacità di calcolo visto l'onere computazionale caratteristico degli LLM. Nei casi in cui non si disponga di queste risorse, è stato dimostrato che sfruttare la rappresentazione *tf-idf* risulta essere una valida alternativa.

Appendice A

Modelli di Classificazione

A.1 Regressione Logistica

La Regressione Logistica (Hastie et al. (2009)) è uno dei modelli parametrici più diffusi per la previsione di variabili binarie. Lo schema della regressione logistica appartiene alla famiglia dei modelli lineari generalizzati (GLM) i quali assumono che la relazione tra Y e le covariate $x_1, \ldots, x_p$ possa essere espressa nella forma:

$$g\left(\mathbb{E}\{Y|x_1, \ldots, x_p\}\right) = \sum_{i=1}^{p} x_i \beta_i$$

dove $g(\cdot)$ è un'opportuna *funzione di legame*.

Nel caso della regressione logistica si assume che il vettore n-dimensionale Y, dato il vettore di covariate x, abbia distribuzione di probabilità binomiale di parametro $\pi(x)$ ($Y \sim Bin(n, \pi(x))$). In questo caso la funzione di legame è:

$$g\left(\pi(x)\right) = \text{logit}\left(\pi(x)\right) = \log\frac{\pi(x)}{1-\pi(x)}$$

formulando la relazione tra π e $x_1, \ldots, x_p$ come:

$$\text{logit}\left(\pi(x)\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

L'assunzione distributiva che viene fatta permette la presenza di una teoria inferenziale basata sulla verosimiglianza e la stima dei parametri avviene tramite la massimizzazione della verosimiglianza binomiale. In generale, però, nell'ambito dei GLM non è possibile esprimere in maniera esplicita la stima di massima verosimiglianza (MLE) ed

è necessario utilizzare procedure di calcolo numerico iterativo come ad esempio l'algoritmo dei *minimi quadrati pesati iterati*.

La procedura per la classificazione binaria che sfrutta questo modello consiste nello stimare, per ciascuna osservazione, la probabilità $\pi(x)$ di appartenere alla classe di riferimento e confrontarla con un valore soglia prefissato k (solitamente 0.5 o calcolato come proporzione campionaria nel caso di dataset sbilanciati). Se $\pi(x) > k$ allora quella osservazione viene classificata come appartenente alla classe di riferimento.

Spesso, specialmente quando la matrice dei regressori X è ad elevata dimensionalità, il modello è soggetto al problema del sovra-adattamento (o *overfitting*) che si manifesta quando un modello di ML si adatta eccessivamente ai dati di stima, perdendo la capacità di generalizzare correttamente su nuovi dati non ancora osservati. Una soluzione comune, utilizzata anche in altre classi di modelli, è la regolarizzazione. L'idea alla base di questa procedura è quella di penalizzare la funzione da ottimizzare (in questo caso la verosimiglianza binomiale) rispetto i parametri in modo tale da diminuirne la variabilità. Indicando la funzione di log-verosimiglianza binomiale come $\ell(\beta; y, x)$, i tipi di penalizzazione più diffuse sono:

- **Lasso (L1)**: si impone il vincolo per cui la somma dei valori assoluti (norma 1) dei coefficienti β sia inferiore ad una soglia λ_{L1} inversamente proporzionale a C:

$$\hat{\pi}(x)_{\text{Lasso}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \left[\ell(\beta; y, x) - \lambda_{L1} \sum_{j=0}^{p} |\beta_j| \right]$$

  Un vantaggio aggiuntivo di questo tipo di penalità è quello di portare il modello ad una selezione automatica delle variabili esplicative in quanto nel processo di stima i coefficienti meno importanti importanti vengono azzerati.

- **Ridge (L2)**: in questo caso il vincolo è che a somma dei quadrati (norma 2) dei coefficienti β sia inferiore alla soglia λ_{L2} ottenendo la formulazione:

$$\hat{\pi}(x)_{\text{Ridge}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \left[\ell(\beta; y, x) - \lambda_{L2} \sum_{j=0}^{p} \beta_j^2 \right]$$

  In questo caso i coefficienti subiscono uno schiacciamento (*shrinkage*) verso lo 0. Ciò, oltre a limitare il sovra-adattamento, gestisce il problema della multicollinearità che si presenta quando le covariate sono fortemente correlate tra loro.

- **ElasticNet**: consiste nella combinazione delle penalità L1 e L2 in un unico modello in modo tale da sfruttare i vantaggi di entrambi i tipi di regolarizzazione. Vengono introdotti quindi due parametri di regolazione, uno per la penalità L1 (λ_{L1}) e uno per la penalità L2 (λ_{L2}), che consentono di bilanciare la componente di selezione delle variabili e la stabilità del modello:

$$\hat{\pi}(x)_{\text{ElasticNet}} = \underset{\boldsymbol{\beta}}{\text{argmax}} \left[\ell(\beta; y, x) - \lambda_1 \sum_{j=0}^{p} |\beta_j| - \lambda_2 \sum_{j=0}^{p} \beta_j^2 \right]$$

Il tipo di penalizzazione usata e relativi parametri sono iperparametri importanti da selezionare per ottimizzare la capacità predittiva del modello.

L'estensione al caso multiclasse della Regressione Logistica che verrà considerato in questo lavoro è uno degli approcci più comuni: "*One-vs-Rest*" (OvR). L'idea alla base consiste nel considerare una Regressione Logistica "classica" separata per ciascuna classe. Per ogni classe, quindi, il modello viene stimato in modo tale da distinguere tale classe da tutte le altre (una classe vs. il resto, da cui il nome "*One-vs-Rest*"). Ogni classificatore binario restituisce una probabilità di "appartenenza" alla classe specifica. Di conseguenza, il modello globale produce un vettore di probabilità con un elemento per ciascuna classe e ogni osservazione, quindi, viene classificata con la classe la cui probabilità associata è più elevata.

A.2 Random Forest

Nei casi come quello trattato in cui la matrice dei regressori X(Hastie et al. (2009)) è caratterizzata da un'elevata dimensionalità che porta facilmente al manifestarsi del sovra-adattamento, una classe di modelli particolarmente indicata per questi casi e che limita per costruzione il problema sono i modelli di *ensembling* (*Ensemble learning*). L'idea alla base delle tecniche di *ensembling* è quella di combinare, attraverso un'opportuna funzione di aggregazione, le previsioni di più modelli di classificazione di base stimati su diversi insiemi di stima per migliorare la precisione e la robustezza delle previsioni finali. Solitamente si considerano come modelli di base classificatori che presentano una significativa componente di variabilità data data dal sovra-adattamento ai dati di stima e quindi fortemente instabili. In questo modo ciascun modello di base cattura pattern specifici del proprio insieme di stima e porta il proprio contributo alla previsione finale nella fase di aggregazione in modo tale da produrre un modello finale che sia in qualche modo in grado di raccogliere le qualità delle singole componenti che lo compongono. Un modello di base molto diffuso in questo ambito e decisamente adatto grazie alla sua

instabilità è l'albero decisionale (*decision tree*) il quale mira a stimare la funzione di probabilità della variabile d'interesse mediante una funzione a gradini.

Il modello di *ensembling* considerato in questo studio è la Foresta Casuale o *Random Forest* (Hastie et al. (2009)), la cui fondazione teorica si basa sui principi di *bootstrap aggregating* (o *bagging*) e di selezione casuale delle variabili (*feature randomization*), uniti all'impiego di un gran numero di alberi decisionali. La procedura di stima di una Foresta Casuale può essere riassunta in diverse fasi:

1. **Bootstrap aggregating (Bagging)**: Per ciascun albero decisionale da stimare, viene creato un campione bootstrap, ovvero un set di dati creato tramite campionamento con reinserimento delle unità osservate nell'insieme di stima originale. Sia $\mathcal{D}_{stima}$ il dataset di stima, B il numero di alberi decisionali di base da stimare e sia M il numero di unità di ciascun campione Bootstrap, in questa fase, verranno costituiti $\mathcal{D}^*_{(1)}, \ldots, \mathcal{D}^*_{(B)}$, insiemi di stima di numerosità M. Il numero di campioni, e quindi di alberi considerati B gioca il ruolo di parametro di regolazione del modello e deve essere scelto in maniera opportuna per minimizzare l'errore di previsione.

2. **Crescita degli alberi decisionali**: i B alberi decisionali vengono stimati attraverso algoritmi iterativi come la fase di crescita dell'algoritmo CART in cui, ad ogni iterazione vengono ispezionati, per ogni variabile, tutti i valori osservati. Viene quindi individuato il valore della variabile (nodo) per cui, se lo si considera come valore soglia per una partizione dei dati, si ha un guadagno massimo in termini di una certa funzione obiettivo. Individuato tale valore, lo spazio dei dati viene quindi diviso costituendo iterativamente nuove regioni sempre più piccole alle quali viene assegnato come valore costante la proporzione campionaria della classe di riferimento per i dati contenuti in quella regione. Questa procedura viene iterata fino a quando il guadagno non si stabilizza. Per una trattazione più esaustiva dell'algoritmo di crescita CART si rimanda a Breiman et al. (1984). La funzione obiettivo generalmente è la devianza D la quale, a seconda della tipologia di problema che si sta affrontando, assume una formulazione specifica. Nel caso della classificazione binaria si considera come devianza la quantità $-2\ell(\theta)$, dove

$\ell(\theta)$ è la log-verosimiglianza binomiale:

$$\begin{aligned} D &= -2\sum_{i=1}^{n}\{y_i \log p_i + (1-y_i)\log(1-p_i)\} \\ &= -2\sum_{j=1}^{J} n_j\{\hat{P}_j \log \hat{P}_j + (1-\hat{P}_j)\log\left(1-\hat{P}_j\right)\} \\ &= 2\sum_{j=1}^{J} n_j Q(\hat{P}_j) \end{aligned}$$

Dove p_i è la probabilità attesa dell'osservazione i di appartenere alla classe di riferimento, $\hat{P}_j$ è la proporzione campionaria di unità appartenenti alla classe di riferimento nella regione j creata in fase di crescita, n_j il numero di unità contenute nella regione j e $Q(\cdot)$ è un indicatore (misura di *impurità*) del fatto che gli elementi di una certa regione (foglia) siano disomogenei rispetto alla variabile risposta. La generalizzazione delle misure d'impurità più utilizzate nel caso della classificazione binaria al caso multiclasse con K classi, è la seguente:

- **Entropia di Shannon**:

$$Q_H(j) = -\sum_{k=1}^{K} P_{jk}\log(P_{jk})$$

- **Indice di Gini**:

$$Q_G(j) = -\sum_{k=1}^{K} P_{jk}(1-P_{jk})$$

Generalmente il CART include anche una fase di "potatura" in cui si vanno ad unire alcune delle J regioni create per limitare il naturale sovra-adattamento della fase di crescita.

3. **Selezione casuale delle variabili**: In realtà, per la stima degli alberi decisionali nelle Foreste Casuali non viene utilizzato direttamente l'algoritmo appena descritto, ma una sua variante in grado di massimizzare i benefici portati dalle procedure di *ensembling*. Il primo aspetto che varia è che, durante ogni iterazione della crescita, non vengo considerati i valori di ogni variabile ma ne viene selezionato casualmente un piccolo sottoinsieme. Lo scopo è quello di ottenere alberi di base con una variabilità ancora più elevata in quanto si introduce maggiore casualità nel processo di stima. Con lo stesso obiettivo, non viene effettuata nessuna fase di potatura portando ciascun albero alla sua massima dimensione; sarà

infatti la procedura di aggregazione degli alberi a regolare il sovra-adattamento. Il numero comune ad ogni albero F di variabili da ispezionare in ogni nodo viene generalmente mantenuto costante ed è anch'esso un importante parametro di regolazione del modello che deve essere selezionato opportunamente.

4. **Aggregazione delle previsioni**: In questa fase, per ogni nuova osservazione, viene predetto il valore della variabile risposta attraverso ogni albero ottenendo il vettore B-dimensionale di predizioni $\hat{y} = (\hat{y}_{(1)}, \ldots, \hat{y}_{(B)})$, dove $\hat{y}_{(i)}$ rappresenta la previsione dell'albero i-esimo.
   La previsione finale sarà data dall'aggregazione tramite voto di maggioranza degli elementi del vettore:
$$\hat{y}_{final} = \text{moda}(\hat{y}_{(1)}, \ldots, \hat{y}_{(B)})$$

La procedura di stima del Random Forest offre diversi vantaggi oltre all'intrinseca gestione del sovra-adattamento. Essa infatti è per sua natura parallelizzabile in quanto ogni albero che compone la foresta è indipendente dagli altri e, considerando un numero di variabili ristretto ad ogni iterazione, la complessità computazionale è limitata rispetto alle altre tecniche di *ensemble-learning*.

A.3 Support Vector Machine

Un altro modello di classificazione di largo utilizzo è la *Support Vector Machine* o SVM (Hastie et al. (2009)). Immaginando le osservazioni x come punti nello spazio p-dimensionale dei regressori, l'idea alla base di questo modello è quella di trovare l'iperpiano ottimale che separa i punti appartenenti a diverse classi in modo da massimizzare la distanza dall'iperpiano m del punto più vicino di ogni classe.

Per comodità assumiamo di essere nel caso della classificazione binaria e che le due classi siano codificate $+1$ e -1. Per un dato iperpiano lineare $\{x : f(x) = \beta_0 + x^\intercal\beta = 0\}$ (in cui si impone $||\beta|| = 1$ per non incorrere in problemi di identificabilità) il problema di ottimizzazione può essere scritto come:

$$\begin{gathered} \max_{\beta_0,\beta} m \\ \text{con vincoli: } ||\beta|| = 1 \ \text{ e } \ y_i(\beta_0 + x_i^\intercal\beta) > m, \quad \text{per } i = 1, \ldots, n \end{gathered} \tag{A.1}$$

Il quale, dopo opportuni passaggi, può essere riscritto nella forma di un problema di minimo per una funzione quadratica con vincoli lineari sullo spazio:

$$\begin{aligned} &\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 \\ \text{con vincoli:} \quad &y_i(\beta_0 + x_i^\intercal \beta) \geq 1, \quad \text{per } i = 1, \ldots, n \end{aligned} \tag{A.2}$$

La soluzione di questo problema può essere ricavata con note procedure di ottimizzazione quadratica che portano alla definizione della regola di classificazione

$$\mathcal{C}(x) = \begin{cases} +1 & \text{se } \hat{\beta}_0 + x^\intercal \hat{\beta} > 0 \\ -1 & \text{se } \hat{\beta}_0 + x^\intercal \hat{\beta} < 0 \\ \text{casualmente} & \text{se } \hat{\beta}_0 + x^\intercal \hat{\beta} = 0 \end{cases}$$

La procedura appena descritta è attuabile però solamente partendo da una forte assunzione sui dati: i due insiemi definiti dalle due classi devono essere tra loro *linearmente separabili* ovvero che ammettano l'esistenza di un'iperpiano che perfettamente li separi.

Nei contesti reali è molto raro trovare situazioni in cui la nuvola dei punti dati dalle osservazioni rispetta la condizione di perfetta separabilità. Risulta quindi impossibile separare i gruppi senza tollerare il fatto che alcuni punti non si troveranno nel lato corretto dell'iperpiano. Un'estensione che permette di affrontare questi casi prevede l'introduzione di variabili ausiliarie a componenti non-negative $\xi = (\xi_1 \ldots, \xi_n)$ che esprimono la distanza di ciascun punto mal classificato dall'iperpiano (se x_i è classificato correttamente $\xi_i = 0$). In questo modo è possibile adattare la formulazione A.1 sostituendo l'i-esimo vincolo $y_i(\beta_0 + x_i^\intercal \beta) > m$ con $y_i(\beta_0 + x_i^\intercal \beta) > m(1 - \xi_i)$ ricavando l'adattamento della A.2 nella forma:

$$\begin{aligned} &\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 + \gamma \sum_{i=1}^{n} \xi_i \\ \text{con vincoli:} \quad &y_i(\beta_0 + x_i^\intercal \beta) \geq 1 - \xi_i \;\text{ e }\; \xi_i \geq 0 \quad \text{per } i = 1, \ldots, n \end{aligned} \tag{A.3}$$

Dove γ è un parametro di regolazione positivo che assume il significato di costo delle violazioni della barriera definita dall'iperpiano. Nel caso linearmente separabile, $\gamma = \infty$. Si può dimostrare che la soluzione del problema di minimo è del tipo:

$$\hat{\beta} = \sum_{i=1}^{n} a_i x_i y_i \tag{A.4}$$

in cui solo alcuni dei pesi a_i sono diversi da 0. Ciò sta a significare che la soluzione è esprimibile in funzione di un gruppo ristretto di osservazioni x_i detti vettori di supporto (*support vectors*).

Nei casi di problemi di classificazione complessi spesso le classi non possono essere separate da un iperpiano lineare, risulta quindi necessario identificare un criterio di separazione più flessibile. A tal scopo la formulazione dell'iperpiano viene sostituita da quella della curva di separazione:

$$\{x : f(x) = \beta_0 + h(x)^\intercal \beta = 0\}$$

dove $h(x) = (h_1(x), \ldots, h_q(x))^\intercal$ è una certa trasformazione delle $x \in \mathbb{R}^p$ variabili esplicative. Considerando $\hat{\beta}$ definito nella A.4 la curva stimata può essere scritta come:

$$\begin{aligned} \hat{f}(x) &= \hat{\beta}_0 + \sum_{i=1}^{n} a_i y_i \, h(x)^\intercal h(x_i) \\ &= \hat{\beta}_0 + \sum_{i=1}^{n} a_i y_i \, \langle h(x), h(x_i) \rangle \end{aligned}$$

in cui $K(x, x') = \langle h(x), h(x') \rangle$ sono funzioni nucleo (o *kernel functions*) definite solo in base al prodotto interno tra i punti ("trucco del kernel"). La funzione nucleo utilizzata è essa stessa un parametro di regolazione del modello e la sua scelta ha spesso un forte impatto sulla capacità predittiva del modello. Non esistono criteri di scelta a priori della funzione più indicata ma essa avviene in genere provando diverse alternative ricavando la configurazione ottimale. Le funzioni nucleo più diffuse sono le seguenti:

- **Polinomiale**: è poco onerosa a livello computazionale e funziona particolarmente bene quando sono presenti relazioni non lineari di grado non troppo elevato nei dati. Si definisce come
$$K(x, x') = (1 + \langle x, x' \rangle)^d$$
il grado del polinomio d è un valore fissato a priori;

- **Base Radiale** (*Radial Basis Function* - RBF): è la funzione *kernel* più diffusa in assoluto grazie alla sua grande flessibilità. Permette di modellare relazioni non lineari molto più complesse rispetto al caso precedente. La sua formulazione è
$$K(x, x') = \exp(-\lambda ||x - x'||^2)$$
dato il parametro di scala λ ;

- **Sigmoidale**: è la funzione più indicata in caso di relazioni complesse asimmetriche e non stazionarie tra le variabili esplicative. In questi pochi casi specifici è preferibile a quello a base radiale, che rimane comunque il più versatile.

$$K(x, x') = \tanh(k_1 \langle x, x' \rangle + k_2)$$

  in questo caso sono due gli iperparametri di regolazione aggiuntivi da scegliere: k_1 e k_2.

L'introduzione di queste funzioni non lineari chiaramente comporta un aumento di complessità del modello ma che, grazie al fatto che le stime si basano solo sui vettori di supporto, non impone un costo computazionale eccessivo.

Appendice B

Risultati dei modelli con parametri di default

B.1 Classificazione binaria

`Siderurgico`

Input	Modello	F1	Recall	Soglia	Tempo
tf-idf	R. Logistica	0.864	0.853	0.723	1.548s
tf-idf	SVM	0.877	0.874	0.755	451.405s
tf-idf	R. Forest	0.845	0.821	0.748	24.422s
tf-idf	Rete Neurale	0.835	0.839	0.853	33.637s
tf-idf + prep.	R. Logistica	0.867	0.869	0.699	1.421s
tf-idf + prep.	SVM	**0.884**	0.903	0.690	401.201s
tf-idf + prep.	R. Forest	0.840	0.811	0.730	43.406s
tf-idf + prep.	Rete Neurale	0.838	0.842	0.875	32.678s
BERTino	R. Logistica	0.827	0.813	0.665	1.030s
BERTino	SVM	0.846	0.835	0.717	136.625s
BERTino	R. Forest	0.826	0.814	0.670	88.539s
BERTino	Rete Neurale	**0.859**	0.860	0.671	40.481s
BERTino + prep.	R. Logistica	0.818	0.821	0.654	1.051s

Tabella B.1: Risultati dataset `Siderurgico`

Input	**Modello**	**F1**	**Recall**	**Soglia**	**Tempo**
BERTino + prep.	SVM	0.839	0.839	0.702	142.193s
BERTino + prep.	R. Forest	0.829	0.846	0.636	86.923s
BERTino + prep.	Rete Neurale	0.853	0.866	0.689	40.042s
LaBSE	R. Logistica	0.848	0.851	0.642	0.672s
LaBSE	SVC	**0.868**	0.868	0.695	124.376s
LaBSE	R. Forest	0.853	0.862	0.640	87.307s
LaBSE	Rete Neurale	0.858	0.852	0.721	32.415s
LaBSE + prep.	R. Logistica	0.821	0.809	0.679	0.721s
LaBSE + prep.	SVM	0.855	0.860	0.679	141.137s
LaBSE + prep.	R. Forest	0.826	0.814	0.665	88.103s
LaBSE + prep.	Rete Neurale	0.856	0.861	0.642	29.750s
MiniLM	R. Logistica	0.804	0.790	0.704	0.364s
MiniLM	SVC	0.843	0.832	0.753	81.198s
MiniLM	R. Forest	0.802	0.796	0.670	55.377s
MiniLM	Rete Neurale	0.852	0.875	0.606	20.125s
MiniLM + prep.	R. Logistica	0.808	0.821	0.663	0.297s
MiniLM + prep.	SVM	**0.860**	0.886	0.673	80.404s
MiniLM + prep.	R. Forest	0.810	0.818	0.667	57.220s
MiniLM + prep.	Rete Neurale	0.836	0.826	0.745	24.416s
BERTino F.T.	R. Logistica	0.885	0.881	0.747	0.592s
BERTino F.T.	SVM	0.887	0.880	0.872	71.301s
BERTino F.T.	R. Forest	**0.889**	0.898	0.647	131.689s
BERTino F.T.	Rete Neurale	0.885	0.885	0.745	27.884s

Tabella B.1: Risultati dataset `Siderurgico`

`Multiservizi`

Input	**Modello**	**F1**	**Recall**	**Soglia**	**Tempo**
tf-idf	R. Logistica	0.781	0.855	0.117	3.372s
tf-idf	SVM	0.805	0.818	0.105	1449.68s
tf-idf	R. Forest	0.785	0.798	0.140	65.374s
tf-idf	Rete Neurale	0.766	0.808	0.000	69.443s
tf-idf + prep.	R. Logistica	0.810	0.823	0.137	3.333s
tf-idf + prep.	SVM	**0.814**	0.820	0.107	1471.898s
tf-idf + prep.	R. Forest	0.805	0.782	0.140	92.413s
tf-idf + prep.	Rete Neurale	0.787	0.769	0.001	69.285s
BERTino	R. Logistica	0.731	0.818	0.119	1.371s
BERTino	SVM	**0.801**	0.729	0.134	357.141s
BERTino	R. Forest	0.739	0.739	0.165	203.062s
BERTino	Rete Neurale	0.778	0.800	0.130	73.998s
BERTino + prep.	R. Logistica	0.765	0.713	0.139	1.999s
BERTino + prep.	SVM	0.800	0.725	0.128	365.794s
BERTino + prep.	R. Forest	0.737	0.707	0.161	194.854s
BERTino + prep.	Rete Neurale	0.771	0.783	0.117	71.423s
LaBSE	R. Logistica	0.769	0.803	0.130	1.697s
LaBSE	SVM	**0.811**	0.754	0.118	347.350s
LaBSE	R. Forest	0.755	0.746	0.162	232.772s
LaBSE	Rete Neurale	0.806	0.790	0.146	72.801s
LaBSE + prep.	R. Logistica	0.788	0.745	0.143	2.137s
LaBSE + prep.	SVM	0.802	0.766	0.110	467.629s
LaBSE + prep.	R. Forest	0.771	0.727	0.171	231.682s
LaBSE + prep.	Rete Neurale	0.803	0.774	0.109	105.806s
MiniLM	R. Logistica	0.775	0.745	0.132	0.866s
MiniLM	SVM	**0.808**	0.768	0.111	292.819s
MiniLM	R. Forest	0.765	0.719	0.162	173.618s
MiniLM	Rete Neurale	0.767	0.800	0.095	50.279s

Tabella B.2: Risultati dataset `Multiservizi`

Input	Modello	F1	Recall	Soglia	Tempo
MiniLM + prep.	R. Logistica	0.798	0.679	0.149	0.647s
MiniLM + prep.	SVM	0.807	0.750	0.114	207.993s
MiniLM + prep.	R. Forest	0.750	0.750	0.156	166.351s
MiniLM + prep.	Rete Neurale	0.780	0.778	0.083	46.714s
BERTino F.T.	R. Logistica	0.770	0.799	0.135	1.455s
BERTino F.T.	SVM	**0.837**	0.736	0.131	299.009s
BERTino F.T.	R. Forest	0.794	0.776	0.176	195.796s
BERTino F.T.	Rete Neurale	0.802	0.821	0.110	62.397s

Tabella B.2: Risultati dataset `Multiservizi`

`Combinato`

Input	Modello	F1	Recall	Soglia	Tempo
tf-idf	R. Logistica	0.845	0.821	0.368	8.733s
tf-idf	SVM	0.854	0.860	0.288	5325.199s
tf-idf	R. Forest	0.835	0.836	0.335	67.870s
tf-idf	Rete Neurale	0.847	0.826	0.200	90.248s
tf-idf + prep.	R. Logistica	0.842	0.842	0.348	6.053s
tf-idf + prep.	SVM	**0.865**	0.838	0.352	5818.643s
tf-idf + prep.	R. Forest	0.835	0.867	0.287	77.947s
tf-idf + prep.	Rete Neurale	0.850	0.820	0.215	99.812s
BERTino	R. Logistica	0.814	0.759	0.387	3.782s
BERTino	SVM	**0.847**	0.765	0.373	1197.851s
BERTino	R. Forest	0.819	0.779	0.393	250.491s
BERTino	Rete Neurale	0.846	0.816	0.364	91.012s
BERTino + prep.	R. Logistica	0.790	0.780	0.335	3.968s
BERTino + prep.	SVM	0.823	0.786	0.312	1144.586s
BERTino + prep.	R. Forest	0.808	0.754	0.403	606.068s

Tabella B.3: Risultati dataset `Combinato`

Input	Modello	F1	Recall	Soglia	Tempo
BERTino + prep.	Rete Neurale	0.823	0.835	0.285	103.642s
LaBSE	R. Logistica	0.804	0.803	0.326	5.416s
LaBSE	SVM	**0.853**	0.822	0.311	1272.726s
LaBSE	R. Forest	0.809	0.782	0.387	356.962s
LaBSE	Rete Neurale	0.852	0.809	0.537	118.207s
LaBSE + prep.	R. Logistica	0.797	0.766	0.355	3.840s
LaBSE + prep.	SVM	0.839	0.816	0.316	1349.118s
LaBSE + prep.	R. Forest	0.804	0.798	0.372	402.146s
LaBSE + prep.	Rete Neurale	0.835	0.814	0.257	112.205s
MiniLM	R. Logistica	0.769	0.803	0.310	1.800s
MiniLM	SVM	**0.849**	0.815	0.351	1065.794s
MiniLM	R. Forest	0.790	0.728	0.398	274.634s
MiniLM	Rete Neurale	0.828	0.810	0.188	83.896s
MiniLM + prep.	R. Logistica	0.781	0.757	0.350	1.593s
MiniLM + prep.	SVM	0.837	0.838	0.305	1037.954s
MiniLM + prep.	R. Forest	0.785	0.765	0.383	238.756s
MiniLM + prep.	Rete Neurale	0.815	0.816	0.249	74.183s
BERTino F.T.	R. Logistica	0.906	0.882	0.320	1.645s
BERTino F.T.	SVM	**0.911**	0.905	0.180	379.451s
BERTino F.T.	R. Forest	0.900	0.881	0.367	424.164s
BERTino F.T.	Rete Neurale	0.910	0.892	0.295	948.903s

Tabella B.3: Risultati dataset `Combinato`

B.2 Classificazione multiclasse

`Farmaceutico`

Input	Modello	F1	Accuracy	Tempo
tf-idf	R. Logistica	0.740	0.745	37.22s
tf-idf	SVC	0.700	0.714	518.877s
tf-idf	R. Forest	0.695	0.709	18.05s
tf-idf	Rete Neurale	0.713	0.718	26.243s
tf-idf + prep.	R. Logistica	**0.745**	0.749	38.22s
tf-idf + prep.	SVC	0.703	0.718	516.418s
tf-idf + prep.	R. Forest	0.727	0.738	18.333s
tf-idf + prep.	Rete Neurale	0.721	0.726	26.498s
BERTino	R. Logistica	**0.682**	0.682	20.864s
BERTino	SVC	0.652	0.677	78.242s
BERTino	R. Forest	0.558	0.59	85.421s
BERTino	Rete Neurale	0.676	0.689	30.179s
BERTino + prep.	R. Logistica	0.659	0.656	20.977s
BERTino + prep.	SVC	0.647	0.672	75.68s
BERTino + prep.	R. Forest	0.563	0.593	87.454s
BERTino + prep.	Rete Neurale	0.652	0.665	26.964s
LaBSE	R. Logistica	0.699	0.697	20.194s
LaBSE	SVC	**0.715**	0.727	79.078s
LaBSE	R. Forest	0.559	0.595	632.129s
LaBSE	Rete Neurale	0.697	0.704	24.103s
LaBSE + prep.	R. Logistica	0.697	0.694	20.677s
LaBSE + prep.	SVC	0.692	0.705	74.421s
LaBSE + prep.	R. Forest	0.549	0.585	82.106s
LaBSE + prep.	Rete Neurale	0.682	0.694	24.361s
MiniLM	R. Logistica	0.639	0.634	9.419s
MiniLM	SVC	**0.669**	0.68	56.69s

Tabella B.4: Risultati dataset `Farmaceutico`

Input	Modello	F1	Accuracy	Tempo
MiniLM	R. Forest	0.528	0.548	68.973s
MiniLM	Rete Neurale	0.631	0.643	19.179s
MiniLM + prep.	R. Logistica	0.644	0.639	9.285s
MiniLM + prep.	SVC	0.662	0.685	56.025s
MiniLM + prep.	R. Forest	0.540	0.561	63.261s
MiniLM + prep.	Rete Neurale	0.661	0.667	18.733s
BERTino F.T.	R. Logistica	0.769	0.773	14.124s
BERTino F.T.	SVC	**0.776**	0.783	28.765s
BERTino F.T.	R. Forest	0.765	0.775	85.115s
BERTino F.T.	Rete Neurale	0.759	0.762	25.551s

Tabella B.4: Risultati dataset `Farmaceutico`

`Multiservizi`

Input	Modello	F1	Accuracy	Tempo
tf-idf	R. Logistica	0.714	0.719	18.425s
tf-idf	SVM	0.663	0.653	169.675s
tf-idf	R. Forest	0.671	0.693	10.762s
tf-idf	Rete Neurale	0.708	0.711	13.005s
tf-idf + prep.	R. Logistica	**0.720**	0.722	15.251s
tf-idf + prep.	SVM	0.666	0.660	184.839s
tf-idf + prep.	R. Forest	0.679	0.699	10.763s
tf-idf + prep.	Rete Neurale	0.718	0.721	16.627s
BERTino	R. Logistica	**0.559**	0.550	8.472s
BERTino	SVM	0.482	0.475	53.726s
BERTino	R. Forest	0.462	0.495	46.700s
BERTino	Rete Neurale	0.540	0.570	13.521s
BERTino + prep.	R. Logistica	0.549	0.543	8.624s

Tabella B.5: Risultati dataset `Multiservizi`

Input	**Modello**	**F1**	**Accuracy**	**Tempo**
BERTino + prep.	SVM	0.489	0.480	48.638s
BERTino + prep.	R. Forest	0.476	0.509	47.456s
BERTino + prep.	Rete Neurale	0.527	0.556	15.635s
LaBSE	R. Logistica	0.625	0.620	8.420s
LaBSE	SVC	0.617	0.607	44.159s
LaBSE	R. Forest	0.506	0.537	45.753s
LaBSE	Rete Neurale	0.612	0.632	13.952s
LaBSE + prep.	R. Logistica	**0.635**	0.631	9.492s
LaBSE + prep.	SVM	0.619	0.619	41.763s
LaBSE + prep.	R. Forest	0.508	0.542	45.717s
LaBSE + prep.	Rete Neurale	0.616	0.628	12.779s
MiniLM	R. Logistica	0.583	0.572	3.772s
MiniLM	SVM	0.607	0.597	26.558s
MiniLM	R. Forest	0.462	0.487	32.457s
MiniLM	Rete Neurale	0.580	0.596	10.257s
MiniLM + prep.	R. Logistica	0.589	0.580	4.043s
MiniLM + prep.	SVM	**0.611**	0.603	26.543s
MiniLM + prep.	R. Forest	0.495	0.522	33.874s
MiniLM + prep.	Rete Neurale	0.600	0.618	13.150s
BERTino F.T.	R. Logistica	**0.823**	0.821	7.379s
BERTino F.T.	SVM	0.804	0.793	20.897s
BERTino F.T.	R. Forest	0.780	0.797	39.502s
BERTino F.T.	Rete Neurale	0.810	0.816	13.778s

Tabella B.5: Risultati dataset `Multiservizi`

Bibliografia

Axelos (2024). Sito ufficiale di axelos. `https://www.axelos.com/certifications/itil-service-management`. Accessed: 2024-06-23.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.

Brown, T. B., Kaplan, J., Mann, B., Dhariwal, P., Ryder, N., Neelakantan, A., Subbiah, M., Shyam, P., Askell, A., Child, R., Agarwal, S., Ramesh, A., Hesse, C., Chess, B., Chen, M., Herbert-Voss, A., Ziegler, D. M., Sigler, E., Clark, J., Krueger, G., Wu, J., Litwin, M., Sastry, G., Henighan, T., Winter, C., Gray, S., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165v1* .

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd ed.

Hugging Face (2023a). Labse.

Hugging Face (2023b). multi-qa-minilm-l6-cos-v1.

Matteo Muffo, E. B. (2023). Bertino: an italian distilbert model. *arXiv preprint arXiv:2303.18121* `cs.CL`.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Parlamento Europeo (2023). Regolamento del parlamento europeo e del consiglio che stabilisce norme armonizzate sull'intelligenza artificiale (legge sull'ia) e modifica alcuni atti legislativi dell'unione. Accessed: 2024-06-23.

Pat SRL (2024). Sito ufficiale di pat srl. `https://pat.eu/`. Accessed: 2024-06-23.

Pennington, J., Socher, R. & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* .

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving language understanding by generative pre-training. Tech. rep., OpenAI.

Vaswani, A., Shazeer, N., Parmar, N. & Uszkoreit, J. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017)*.

Wikipedia (2024). Natural language processing. `https://en.wikipedia.org/wiki/Natural_language_processing`. Accessed: 2024-06-23.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V. & Norouzi, M. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144v2* .