Università Del Salento

Corso di Laurea Triennale in Ingegneria dell'Informazione

Tesi di Laurea in Ingegneria dell'Informazione

Utilizzo del Natural Language Processing per il rilevamento delle PII

Relatore
Prof. Luca Mainetti

Laureando
Andrea Elia
Matricola n° 20064410

ANNO ACCADEMICO 2023/2024

Indice

Capitolo 1

Introduzione

Negli ultimi anni, la protezione dei dati personali è diventata una priorità assoluta per garantire la sicurezza e privacy delle informazioni sensibili.
L'evoluzione dei servizi digitali, sempre più diffusi, espone enormi quantità di dati contenenti informazioni personali e garantire la loro sicurezza è una delle sfide più significative per istituzioni, aziende e ricercatori.

In questo contesto, il rilevamento automatico delle **Personal Identifiable Information(PII)** ha acquisito un ruolo cruciale.
Identificare automaticamente le PII è diventato fondamentale per garantire la conformità alle normative sulla privacy, dettate dal General Data Protection Regulation(GDPR).
Queste regolamentazione impongono standard rigorosi per la gestione dei dati personali, richiedendo delle tecniche avanzate per l'identificazione e la protezioni di dati sensibili.

1.1 Stato dell'arte

I progressi del **Natural Language Processing (NLP)** hanno reso possibile lo sviluppo di modelli pre-addestrati tra cui BERT (Bidirectional Encoder Representation From Transformer).
BERT, grazie alla sua architettura basata sui Transformer, ha inaugurato una nuova era nell'uso di algoritmi *machine learning* per compiti specifici con una significativa riduzione dei tempi di addestramento e a un miglioramento delle performance.

L'introduzione dei Transformer, ha portato allo sviluppo di modelli ancora più avanzati come RoBERTa, DeBERTa, GPT, ELECTRA che permettono di essere applicati per una vasta gamma di compiti specifici.
Questi modelli rappresentano ancora oggi strumenti fondamentali per l'analisi del linguaggio naturale.

La tecnologie utilizzate nella seguente tesi, per effettuare un rilevamento delle PII, includono PyTorch, Cuda e TensorFlow, selezionate per ottimizzare il processo di addestramento e riduzione di calcolo, sfruttando a pieno la potenza delle GPU.

1.2 Obiettivo della tesi

L'obiettivo principale della tesi è valutare **le capacità di un modello BERT pre-addestrato per il rilevamento delle PII** nel campo del Natural Language Processing(NLP), grazie ad un addestramento ottimale basato su dataset selezionati comprendenti dati di ottima qualità.
Attraverso l'utilizzo di tecniche di fine-tuning e varie tecniche di Cross Validation, si intende verificare se BERT può competere con altri modelli nel rilevamento delle PII.

Inoltre verranno effettuate comparazione con modelli di ultima generazione, come GPT-4 ed altri modelli evoluti di BERT, come CodeBERT.
Per fare queste comparazioni, sono state effettuate delle tecniche di suddivisione del dataset come, **Training-Validation-Test, K-Fold e Stratified K-Fold,** per migliorare la robustezza del modello durante il processo di addestramento.
Tali tecniche verranno messe a confronto, tramite le metriche di **precisione, accuratezza, sensibilità ed F1-Score,** parametri universali per analizzare i comportamenti dei modelli machine learning.

1.3 Struttura della tesi

La tesi è strutturata nei seguenti capitoli:

- **Capitolo 2: Background**- questa sezione introduce il NLP, una disciplina che si occupa dell'interazione tra computer e il linguaggio umano. Viene introdotta la storia dei modelli di machine learning, analizzando come essi abbiamo rivoluzionato il campo.
  Con un'analisi dettagliata dell'architettura di un Transformer, si descrivono le sue componenti ed il suo funzionamento, informazioni necessarie per comprendere la tesi e per capire come il modello si comporta nel rilevamento delle PII.
  Inoltre, si spiega dettagliatamente il processo di *self-attention*, una tecnica adottata dai Transformer che ha rivoluzionato il campo del machine learning, dando particolare attenzione alle componenti chiave di questo processo, quali le matrici di **Attention Score e Attention Weight**.

- **Capitolo 3: Descrizione della soluzione proposta**- In questo capitolo ci focalizziamo sulla definizione di informazioni personali (PII) e sull'utilizzo dei modelli basati su architetture Transformer. Grazie ai paper pubblicati dai vari team di ricerca, si effettua una comparazione tra RoBERTa , ELECTRA ,GPT-3 e BERT. L'analisi ha portato alla conclusione che BERT rappresenta un'opzione versatile per il rilevamento delle PII.
  Inoltre si analizza, tramite i risultati dei principali benchmark per modelli machine-learning, che BERT, osservato nei compiti di classificazione e comprensione del testo, è quello con le performance più promettenti.

- **Capitolo 4: Progettazione**- si fornisce una descrizione approfondita del modello BERT, attraverso l'analisi dei suoi meccanismi di pretraining e fine-tuning. Si analizza nel dettaglio come esso funziona e su quali task si è basato il suo addestramento, quali **Masked Language Model(MLM)** e il **Next Sentence Prediction(NSP)**.
  Inoltre si descrive come effettuare una formattazione adeguata dei dati per l'addestramento, la scelta del modello pre-addestrato e l'analisi dell'architettura per il task di **classificazione binaria**.

- **Capitolo 5: Implementazione**- viene descritta come è stata effettuata l'implementazione del modello in **Python**. Il linguaggio di programmazione Python, grazie al supporto di tecnologie adatte per l'ottimizzazione

del processo di addestramento, è risultato essere quello più ottimale per l'obiettivo della tesi.
Si introduce il codice utilizzato e la sua suddivisione in 4 classi: *BertModelTrainer, DataHandler, PIIDataLoader e PIIdataset.* Strutturato in tale modo per una chiara suddivisione dei compiti e una corretta manutenzione del codice.
Inoltre viene suggerito un esempio di implementazione per l'addestramento tramite la tecnica Training-Validation-Test.
Per consultare il codice completo si rimanda alla repository GitHub[1]

- **Capitolo 6: Validazione**- il capitolo più corposo della tesi, in cui si descrivono i diversi approcci impiegati per l'addestramento del modello BERT, con un'analisi dettagliata alle metodologie utilizzate, quali Training-Validation-Test e le tecniche di Cross Validation come K-Fold e Stratified K-fold.
  Il capitolo inizialmente introduce i concetti di dati di addestramento, dati di validazione e di testing. Vengono descritti inoltre i dataset utilizzati per l'addestramento e il testing, con particolare attenzione data alla qualità di essi, per evitare situazioni di *underfitting o overfitting.*
  Vengono analizzati i risultati degli 11 modelli ottenuti tramite le diverse tecniche, tra cui uno ottenuto tramite Training-Validation-Test, cinque con K-Fold e cinque con Stratified K-Fold.
  Questi modelli vengono comparati tramite la metrica di precisione per analizzare quale tecnica è la più efficace.
  Si evince che Training-Validation-Test è la tecnica più affidabile(anche se di pochi punti percentuali), grazie alla fase di validazione , non presente nelle altre, che permette al modello un'ottimizzazione delle scelte senza la modifica dei suoi gradienti.
  Dopo aver analizzato ciò, si prendono i 3 modelli con prestazioni più elevate, uno per ogni tecnica, e si comparano con altri presenti in letteratura , per osservare il posizionamento del nostro lavoro. Premettendo che tale comparazione è stata effettuata con modelli addestrati con dataset differenti, si vedrà che i nostri modelli avranno metriche superiori rispetto agli altri presi in considerazione

[1] https://github.com/andreaeliia/BertModelPII

Capitolo 2

Background

2.1 Natural Language Processing

Il **Natural Language processing (NLP)** è subdisciplina dell'Intelligenza artificiale che si occupa dell'interazione tra elaboratori e linguaggio umano attraverso la comprensione e la generazione del testo.

Il **NLP** ha diverse applicazioni in numerosi settori, tra cui la finanza, la medicina, l'ambito legale ed ovviamente è diffuso in diverse applicazioni informatiche.
Un esempio di rilevanza è l'utilizzo di **BERT** da parte di Google: dal 2019 viene utilizzato come loro motore di ricerca per il completamento delle frasi.

I task del NLP possono essere suddivisi in due macro gruppi principali:

- **Task linguistici**: riguardanti la comprensione del contesto, il riconoscimento delle entità nominate, il tagging delle parole (ad esempio, identificare un termine come sostantivo o verbo) e l'analisi semantica del testo. Questi compiti mirano ad estrarre e interpretare il significato linguistico e semantico del testo.

- **Task di supporto all'utente**: includono il riconoscimento vocale, la generazione di linguaggio naturale (Natural Language Generation, NLG) e la comprensione del linguaggio naturale (Natural Language Understanding, NLU). Questi compiti sono orientati a migliorare l'interazione uomo-macchina, facilitando la comunicazione naturale e l'interazione con gli utenti.

Nel corso degli anni sono emersi diversi approcci riguardo il NLP.
Inizialmente l'attenzione era rivolta su metodi basati su modelli statistici, i quali attraverso tecniche matematiche e probabilistiche, erano utilizzati per modellare ed analizzare il linguaggio.
Questo approccio, iniziato negli anni '90, si basava su modelli a N-grammi o modelli di Markov nascosti.
Tali modelli, attraverso l'utilizzo del tagging e della segmentazione del testo, prevedevano la probabilità della parola successiva basandosi sulle n-1 parole precedenti.
Tuttavia, nonostante la loro iniziale diffusione, questi modelli sono stati gradualmente superati da tecniche più avanzate, per le loro limitazioni legate alla comprensione del testo, in quanto essi non erano abili nel comprendere le dipendenze tra le varie parole a lungo termine.

Successivamente l'approccio si è evoluto verso l'utilizzo di reti neurali, modelli ispirati al cervello umano che simulano i neuroni e le loro connessioni.
Si basano sulla capacità di riconoscimento del pattern del testo e della sua comprensione, essendo in grado di apprendere in modo più efficace le relazioni e le dipendenze nel linguaggio naturale, accantonando così definitivamente i modelli statistici.

Le reti neurali hanno costituito la base per l'evoluzione del deep learning e il loro impiego ha portato a significativi miglioramenti per il NLP.

Le due principali architetture di reti neurali che hanno avuto una rilevanza in questo campo sono:

- **Reti neurali convoluzionali**: originariamente utilizzate per il riconoscimento delle immagini per estrapolare le loro caratteristiche.
- **Reti neurali Transformer**: introdotte nel 2017 hanno rivoluzionato il NLP, adottando dei meccanismi di attenzione che permettono una comprensione più profonda per il riconoscimento delle dipendenze nel testo.

2.2 Transformer

I **Transformer**, introdotti da google nel 2017, nel paper *Attention Is All You Need* (Vaswani et al., 2017), rappresentano una nuova architettura di rete neurale basata sul meccanismo di auto-attenzione particolarmente adatto alla comprensione del testo [1].

Prende piede, per i task di traduzione di lingue, in quanto rispetto alle reti neurali precedenti (reti neurali ricorrenti e convoluzionali), presenta una quantità minore di calcoli, accelerando così i tempi di addestramento.

Le reti neurali precedenti ai Transformer elaboravano il linguaggio generando rappresentazioni di spazi vettoriali di lunghezza fissa o variabile.
In seguito dopo aver iniziato con rappresentazioni di singole parole, aggregano le informazioni delle parole limitrofe per determinare il contesto della frase.

I Transformer a loro volta, eseguono un numero piccolo e costante di passaggi, in cui in ogni passo si applica un meccanismo di **auto attenzione ("self-attention mechanism")** che modella le relazioni tra le parole, indipendentemente dalla loro posizione nella frase.
Il modello può essere addestrato per far maggiore attenzione ad una singola parola della frase, permettendo di capire il contesto del discorso, riuscendo a distinguere gli omografi con il loro relativo significato.

Per capire la rappresentazione successiva della parola la confronta con le altre componenti della frase.
In seguito il modello assegna un attention score dopo aver effettuato i vari confronti.

Cercando di capire al meglio il suo funzionamento,si riporta un esempio di traduzione, presente nel paper, eseguito da un modello basato sui Transformer.

"I arrived at the bank after crossing the river"

"Sono arrivato alla riva dopo aver attraversato il fiume"

In inglese la parola *bank* può significare sia “banca” che “riva”.
Il modello analizza la parola “river” e la confronta con tutte le parole della frase: una volta arrivato ad analizzare “bank” il modello potrebbe scegliere di rappresentarlo come banca o come sponda. In seguito gli assegna un **attention score** che viene utilizzato come media ponderata per scegliere la parola adatta alla traduzione: la scelta ricadrà nella parola con un più alto attention score.

2.2.1 Architettura di un Trasformer

L’architettura di un Transformer è composta da due componenti principali.

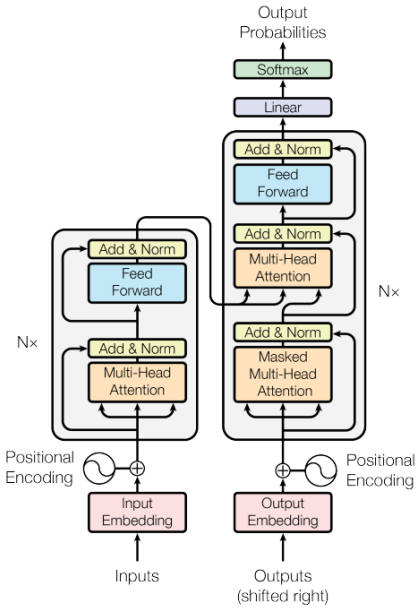

Figura 2.1: Architettura di un Transformer[1]

- **Encoder**: prende in input una sequenza di dati e la trasforma in un vettore di contesto che contiene tutte le informazioni essenziali, in un formato utilizzabile dal Decoder.
- **Decoder**: prende in input il vettore di contesto precedentemente generato dall’Encoder e lo trasforma in una sequenza di output.

2.2.2 Componenti dell'architettura del Transformer

Descriviamo adesso le varie componenti dell'architettura software di un Transformer[2] [3].

Encoder

1. Input Embeddings
L'Encoder converte le parole in input in vettori utilizzando livelli di **embedding ("incorporamento")** che catturano il significato semantico: ogni parola è associata ad un vettore. I vettori possiedono una dimensione fissa di 512. Questa rappresentazione permette al modello di catturare le varie sfumature del significato della parola[3].

2. Positional Encoding
In questo blocco si fanno uso dei vettori posizionali, per far comprendere al modello la posizione di ogni singola parola all'interno della frase. Si utilizzano delle combinazioni di varie funzioni coseno e seno per creare i vettori posizionali [2].

3. Stacks Encoder Layers
Il modello è costituito da una pila di strati identici (variabile in base al modello preso in analisi): hanno lo scopo di trasformare tutte le sequenze di input in una rappresentazione continua e astratta che incapsula tutte le informazioni apprese dall'intera sequenza.

Questo strato è costituito da due sottomoduli principali:

- **Multi-Headed Self-Attention Mechanism**
- **Rete Completamente Connessa**

Nel **Multi-Headed Self-Attention Mechanism** viene utilizzato un meccanismo di self-attention che instaura le diverse relazioni tra le parole componenti la frase. Elabora un **attention score** (2.2.3) che si basa su:

- **Query**: vettore che rappresenta una parola o un token.
- **Key**: vettore corrispondente a ciascuna parola o token.
- **Value**: associato a una key e utilizzato per costruire l'output dell'attention score. Quando una query e una key hanno un alto punteggio, il valore corrispondente viene enfatizzato nell'output.

Ogni sottolivello è seguito da un processo di normalizzazione che aiuta il modello a convergere più rapidamente durante il processo di addestramento.

4. Output of the Encoder
L'output normalizzato della connessione residua passa attraverso una rete neurale **feed-forward puntuale**.
Questa rete è composta da due strati lineari, separati da una funzione di attivazione lineare, generalmente una **ReLU(Rectified Linear Unit)**.
Questo passaggio ha lo scopo di aumentare la capacità del modello di apprendere ed elaborare informazioni in modo più astratto.
In seguito, l'output viene aggiunto all'input della rete stessa tramite una connessione residua, una tecnica per facilitare il flusso del gradiente di scomparsa (**vanishing gradient**) durante l'addestramento[3].
Esso ha la funzione di migliorare la previsione del modello, in quanto nei modelli più profondi(Deep Neural Networks) i gradienti calcolati diventano molto piccoli man mano che attraverso gli strati dell'architetettura Transformer. In seguito l'output viene normalizzato per migliorare l'efficienza del modello.

Decoder

Il **Decoder** si compone di varie componenti simili all'Encoder, ma adattate per gestire gli output generati dall'Encoder.

1. Output Embedding
Analogamente all'input Embedding dell'Encoder, l'**Output Embedding** converte i token di output in vettori che sono processati dal Decoder.

2. Positional Encoding
Simile all'Encoder, il Decoder utilizza questa componente per incorporare informazioni sulla posizione relativa dei token all'interno della sequenza di output.

3. Stack of Decoder Layer

Il Decoder è composto da una serie di strati identici, ognugno dei quali contiene dei sottolivelli.

3.1 Masked Self-Attention Mechanism

È similare al processo di self-attention dell'encoder, tuttavia impedisce al vettore di posizione di cambiare l'ordine.

Questo fa sì che ogni parola non venga influenzata dai token futuri.

3.2 Encoder-Decoder Multi-Head Attention or Cross Attention

Avviene l'interazione tra Decoder e Encoder: l'output generato dall'encoder funge da query e da key, invece gli output generati nel processo di self-attention vengono utilizzati come Value. In questo processo si enfatizzano le parti più importanti della frase, comprendendo il contesto del discorso.

3.3 Feed-Forward Neural Network

Il **Feed-Forward Neural Network** è struttrato come una successione di due strati completamente connessi, separati da un funzione di attivazione non lineare.

È progettata per facilitare una trasformazione profonda delle rappresentazioni di input.

Consiste da due trasformazioni lineari con un'attivazione ReLU nel mezzo[1].

4. Linear Classifier and SoftMax for Generating Output Probabilities

È composto da due strati: **Linear** e **Softmax**. Il Linear funge da classificatore, con una dimensione che corrisponde al numero totale di classi.

Le classi coinvolte sono relative alle parole utilizzate dal modello nel suo vocabolario di termini appresi. In seguito, l'output viene mandato allo strato di SoftMax, che indicizza un intervallo di probabilità (compreso tra 0 e 1): il modello prenderà la parola con la probabilità più alta in modo che sappia predire la sequenza successiva [3].

2.2.3 Attention Score, Softmax e Output

Calcolo dell'attention score
L'attention Score viene calcolato per determinare la rilevanza di ogni parola rispetto alle altre parole presenti nella sequenza. Il calcolo viene eseguito utilizzando le matrici **Query(Q)**, **Key (K)** e **Value (V)**.

L'attention score si ottiene tramite il prodotto della matrice Q e la trasposta della matrice K dividendo per la radice quadrata della dimensione della matrice K.

$$\text{Attention score} = \frac{QK^T}{\sqrt{d_k}} \tag{2.1}$$

Il risultato di questo calcolo sarà una matrice.
Le sue componenti saranno chiamate **logit**(valori grezzi): rappresentano l'affinità tra una Query ed una Key.

Supponiamo di avere una frase con 4 parole "Il gatto ama giocare".
La matrice di attention score avrà una dimensione 4x4, con indici i (indice della parola query) e j (indice della parola key).

$$\text{Attention Score} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Ogni componente di una riga o di una colonna rappresenta il valore della correlazione con una parola all'interno della frase, come illustrato in figura.

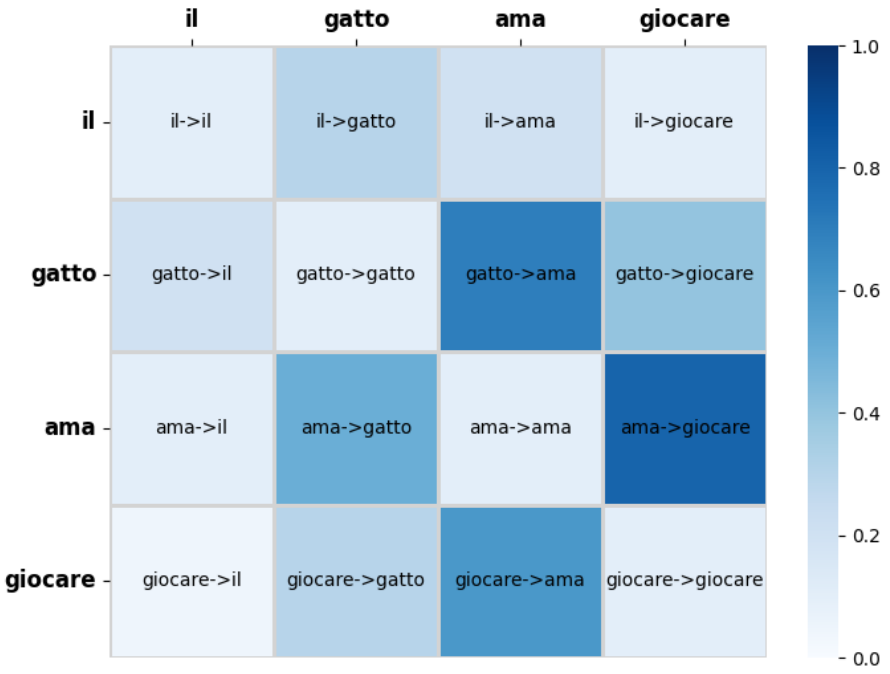

Figura 2.2: Rappresentazione dell'Attention Score e delle sue relazioni

Applicazione del SoftMax

Una volta calcolato l'attention score, il passo successivo consiste nell'applicare la funzione di Softmax a ciascuna componente della matrice.
È fondamentale in quanto si va a normalizzare i valori di logit, cioè si vanno a trasformare i punteggi grezzi in probabilità.
La matrice composta da tali probabilità sarà chiamata **Attention Weight** e determina il grado di correlazione di ciascuna parola rispetto ad una particolare query.

$$\text{Attention Weight} = \text{Softmax}(\text{Attention Score}) \tag{2.2}$$

Calcolo dell'output

In seguito va eseguito il calcolo dell'output per il Self-Attention Mechanism. È calcolato attraverso il prodotto matriciale tra le matrici Attention Weight e Value.

$$\text{Output} = \text{Attention Weight} \times V \tag{2.3}$$

Capitolo 3

Descrizione della soluzione proposta

In questo capitolo analizziamo la definizione di **Personal Identifiable Information(PII)** per definire i vari campi su cui dovremmo addestrare il modello. Inoltre verranno descritti i vari modelli facenti uso di Transformer, motivando la scelta dell'utilizzo di **BERT**, analizzando le prestazioni e i vari casi d'utilizzo descritti nei vari paper.

3.1 Personal Identifiable Information

Le **Personal Identifiable Information (PII)** sono tutte le informazioni collegate a uno specifico individuo che possono essere utilizzate per scoprirne l'identità.

Per classificare tutti i tipi di dati personali, facciamo riferimento al *Regolamento sulla Protezione dei Dati (GDPR)* in particolare all'articolo 4 e 9.

Articolo 4

"dati personali" significa qualsiasi informazione riguardante una persona fisica identificata o identificabile ("interessato"); una persona fisica identificabile è una persona che può essere identificata, direttamente o indirettamente, in particolare facendo riferimento a un identificativo come un nome, un numero di identificazione, dati di localizzazione, un identificativo online o a uno o più elementi specifici, propri dell'identità fisica, fisiologica, genetica, psichica, economica, culturale o sociale di quella persona.

Articolo 9

È vietato il trattamento dei dati personali che rivelano l'origine razziale o etnica, le opinioni politiche, le convinzioni religiose o filosofiche, l'appartenenza sindacale, il trattamento di dati genetici, i dati biometrici finalizzati a identificare in modo univoco una persona fisica, i dati relativi alla salute o alla vita sessuale o all'orientamento sessuale di una persona fisica, salvo che si applichino le condizioni di cui al paragrafo 2."

Comprendo quindi il concetto di informazioni personali, possiamo fare una lista dei possibili campi che possono essere considerati PII.

- Nome completo
- Indirizzo email personale
- Numero di telefono personale
- Indirizzo postale
- Data di nascita
- Codice fiscale
- Numero di patente di guida
- Numero di passaporto
- Informazioni finanziarie: numero carta di credito o IBAN
- Orientamento sessuale
- Opinioni politiche

Da precisare che spostandosi in altri contesti questa definizione tende a modificarsi ed a aggiornarsi. È quindi di vitale importanza capire il contesto in cui ci troviamo per definire i vari campi che possono essere considerati informazioni personali.

3.2 Modelli facenti uso di Transformer

Analizziamo adesso i vari modelli che fanno uso di Transformer, facendo particolare attenzione alle performance di addestramento e al loro utilizzo, guardando i paper pubblicati dalle varie aziende e laboratori di ricerca.

3.2.1 RoBERTa

RoBERTa è un modello evoluto di BERT, introdotto nel 2019 da Facebook AI.
È stato sviluppato attraverso un processo di pre-training ottimizzato rispetto a BERT con un ampio dataset di oltre 160 GB da composto da diverse fonti tra cui BookCorpus (16 GB), CC-News (76 GB), OpenWebText (38 GB) e Stories (31 GB)[4].
Attraverso l'utilizzo di **GLUE** (General Language Understanding Evaluation), benchmark usato per valutare le varie prestazioni dei modelli NLP, ha stabito vari record in 4 dei 9 task effettuati.

- **MNLI**(Multi-Genre Natural Language Inference)
- **QNLI**(Question Natural Language Inference)
- **RTE**(Recognizing Textual Entailment)
- **STS-B**(Semantic textual SImilarity Benchmark)

La fase di pre training ha una durata di addestramento nettamente maggiore rispetto agli altri modelli, un utilizzo di batch di dimensioni maggiori, un dataset più ampio e l'implementazione di un mascheramento dinamico.
Inoltre, ha migliorato le sue prestazioni eliminando l'obiettivo di previsione della frase successiva (NSP).

3.2.2 ELECTRA

Introdotto da Kevin Clark nel 2020, **ELECTRA** si concentra principalmente sul Discriminative Masked Language Modeling.
Questo approccio prevede l'uso di due modelli Transformer: il **generatore** e il **discriminatore**.
Il generatore sostituisce i token in una sequenza con token casuali, mentre il

discriminatore è incaricato di identificare quali token sono stati sostituiti dal generatore[5].

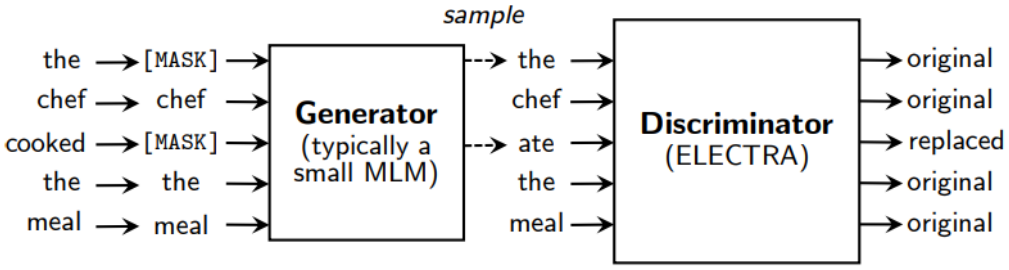

Figura 3.1: Panoramica del funzionamento dei token sostituiti[5].

Il principale vantaggio di ELECTRA risiede nelle sue prestazioni computazionali notevolmente ridotte.
Dai grafici forniti nella documentazione, emerge che ELECTRA mostra un'efficienza superiore rispetto a BERT e RoBERTa nel pre training secondo il benchmark di GLUE.
Tuttavia ELECTRA non è generalmente raccomandato per la fase di fine-tuning, poiché in specifici contesti, si è osservato un calo significativo delle prestazioni rispetto ad altri modelli.

3.2.3 GPT-3

GPT-3, introdotto nel 2020 da OpenAI, è un modello di linguaggio autoregressivo.
È stato addestrato utilizzando il **Next-Word Prediction**, una task non supervisionata che si basa sulla previsione della parola successiva in una frase.
Particolare attenzione va data a questo modello per via dell'utilizzo di solo Decoder all'interno della sua architettura.

La scelta di utilizzare solo Decoder da luogo ad alcuni vantaggi e problematiche allo stesso tempo: essendo composto da **96 Decoder** è molto versatile, in quanto può essere usato per una varietà di compiti legati al NLP.
Tuttavia la sua versatilità fa in modo che non sia particolarmente adatto per un compito specifico, in questo caso la rivelazione delle PII.
Si vedrà in seguito che le sue performance sono inferiori rispetto ad altri modelli. Inoltre nel capitolo 6, relativo agli score e alla comparazione dei modelli,

vedremo che GPT-4 riesce a comprendere, con un tasso particolarmente elevato, le informazione personali, tuttavia non riesce a distinguare in modo ottimale le frasi che non hanno PII al suo interno.

3.2.4 Perché l'utilizzo di BERT?

Nonostante BERT sia il modello meno raccomandato, è stato scelto perché è un compromesso tra i 3 modelli precedentemente elencati.
Questa decisione è stata motivata dall'obiettivo specifico di effettuare un addestramento focalizzato sull'individuazione di informazioni personalmente identificabili (PII), poiché BERT risulta essere il più versatile per tale compito.

3.3 Utiizzo di BERT per il rilevamento delle PII

BERT (Bidirectional Encoder Representation From Transformer), è stato introdotto nel 2018 da Google, è un modello di apprendimento automatico basato sui Transformer, successivamente implementato per la ricerca sul motore di Google.
È stato addestrato con un set di dati di oltre 3.3 miliardi di parole, di cui 2.5 miliardi provenienti da Wikipedia e 800 milioni da Google Books Corpus.

Dall'introduzione di BERT è stato rivoluzionato il mondo del NLP, migliorando le diverse performance negli 11 più comuni compiti dei vari modelli, tra cui:

1. Classificazione del testo
2. Named Entity Recognition (NER)
3. Risposta e domanda (Question and Answering)
4. Sommario Testuale
5. Rilevamento dello Spam
6. Riconoscimento delle relazioni
7. Risposta a domande a scelta multipla

8. Riconoscimento delle intenzioni
9. Analisi del Sentiment
10. Correzione grammaticale
11. Risoluzione di Co-Referenze

In questo caso andremo ad effettuare una **classificazione del testo binaria**.

3.3.1 Prestazioni

Come già accennato nello scorso paragrafo **(3.2.4)**, la scelta dell'utilizzo di BERT è stata accentuata dalle sue prestazioni nei task linguistici.

Nel paper redatto da vari dipendenti di Google *"Universal Language Model Fine-tuning for Text Classification (Howard e Ruder, 2018)*[6], sono state valutate le prestazioni nei vari task linguistici dei modelli ***ELMo, OpenAI GPT,BiLSTM+ELMO+Attn,Pre-OpenAi SOTA, CVT, CSE***, messi a confronto con BERT (BERTbase e BERTlarge), tramite i principali benchmark **GLUE Benchmark e SQuAD v1.1**.

GLUE Benchmark

GLUE (General Language Understanding Evaluation) costituisce uno dei benchmark più utilizzati per valutare le prestazione dei modelli NLP su nove compiti diversi riguardanti la comprensione del linguaggio.

Nel paper di riferimento è stato analizzati su 8 compiti diversi:

- **MNLI (Multi-Genre Natural Language Inference)**: valuta le capacità di inferenza testuale, cioè determina le relazioni tra una coppia di frasi. La seconda frase presa in analisi si riferisce ad un'ipotesi che può assumere significati positivi, neutri o contraddittori.
- **QQP (Quora Question Pairs)**: valuta se le due domande sono semanticamente equivalenti.
- **QNLI (Question Natural Language Inference)**: determina se un'ipotesi è supportata da un contesto dato.

- **SST-2 (Stanford Sentiment TreeBank)**: classificazione del sentiment delle frasi come positivo o negativo.
- **CoLA (Corpus Linguistic Acceptability)**: determina l'accettabilità grammaticale delle frasi.
- **MRPC (Microsoft Research Paraphrase Corpus)**: riconoscimento delle frasi parafrasate.
- **RTE (Recognizing Textual Entailment)**: determina la relazione di implicazione tra frasi.

Dai risultati si nota fin da subito che BERT nei task linguistici ha prestazioni superiori rispetto agli altri modelli.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figura 3.2: Risultati dei test GLUE[6]

SQuAD v1.1 (Stanford Question Answering Dataset)

Creato dall'università di Stanford, è un benchmark per la valutazione delle capacità di comprensione del linguaggio naturale nei modelli di apprendimento automatico. Il dataset comprende 100.000 coppie di domande e risposte, generate da articoli di Wikipedia. I risultati sono in termini di EM[1] (Exact Match) e F1 score[2], sia per il set di sviluppo (Dev)[3], sia per il set di test[4].

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Figura 3.3: Risultati dei test GLUE[6]

[1]Exact Match: percentuale di risposte esatte

[2]F1 Score: combinazione tra precisione e richiamo

[3]Set di sviluppo: dati utilizzati per sintonizzare e migliorare il modello in fase di sviluppo

[4]Set di test: dati non utilizzati durante l'addestramento del modello. Garantiscono l'imparzialità dell'efficacia del modello

Capitolo 4

Progettazione

In questo capitolo viene descritta l'architettura del modello BERT, analizzando il suo funzionamento e illustrando infine la scelta del modello pre-addestrato più idoneo per l'obiettivo della tesi.

4.1 Architettura BERT

L'architettura di BERT è composta da 12 o 24 **Encoder**, in base all'utilizzo di **BERTbase** o **BERTlarge**.

Si usano esclusivamente Encoder per le seguenti ragioni:

- **Comprensione bidirezionale**: grazie al Multi-Headed Self-Attention Mechanism, tutte le parole della frase vengono messe a confronto, sia con quelle precedenti che con quelle successive.
  Utilizzando in aggiunta i Decoder ciò non avverrebbe, in quanto è presente il Mask Self-Attention che non presenta la previsione delle parole successive.

- **Efficienza computazionale e scalabilità**: l'utilizzo degli Encoder semplifica l'architettura del modello e ne migliora l'efficienza computazionale: è facilitata la parallelizzazione del calcolo grazie anche all'elaborazione in Batch.

- **Pre-training**: durante il processo di pre-training, BERT sfrutta due compiti specifici, il *Masked Language Model (MLM)*[1] e il *Next Sentence*

[1]Masked Language Model (MLM): alcune parole della frase sono mascherate. Il modello deve predire correttamente le parole mancanti.

Prediction (NSP)[2], compiti in cui è necessario capire il contesto della frase, e dunque gli Encoder, grazie alla loro bidirezionalità, sono adatti per questo compito.

Per capire al meglio il funzionamento di BERT, ci soffermiamo sulle due fasi principali del modello: il **Pre-training** e il **Fine tuning**.

4.2 Pre Training

Durante la fase di pre-training BERT svolge due compiti simultaneamente per avere una comprensione del linguaggio e del contesto: il **Masked Language Model (MLM)** e **Next Sentence Prediction(NSP)**
. Il pre-training serve per fare comprendere al modello "cosa è un linguaggio?" e "cosa è un contesto?".

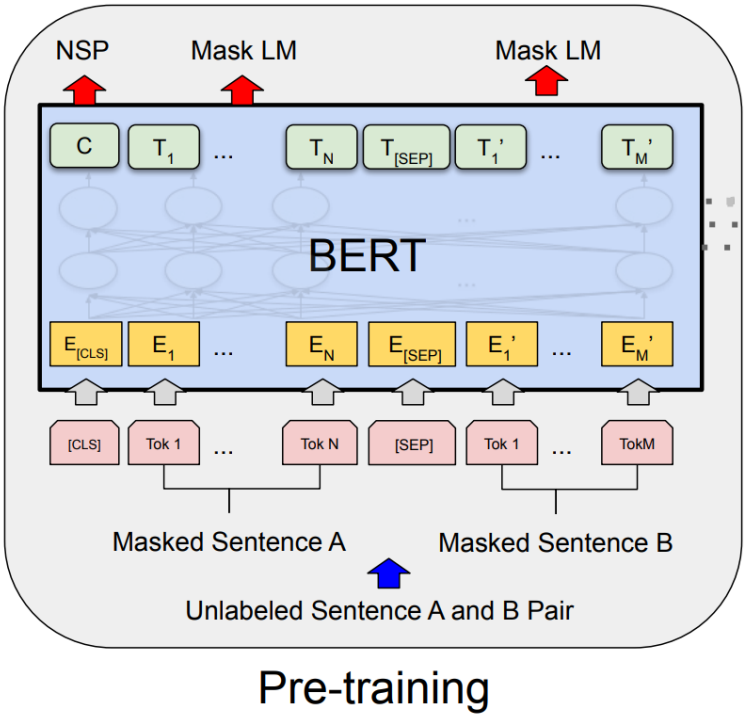

Figura 4.1: Funzionamento del modello BERT.

[2]Next Sentence Prediction (NSP): il modello deve comprendere se due frasi sono logicamente collegate

Per il passaggio dei dati di input, si rappresentano le parole in una sequenza, utilizzando 3 tipologie diverse di embedding.

1. **Token Embedding**: vengono inseriti i token [CLS] all'inizio e [SEP] alla fine della frase.
2. **Segment Embedding**: aiutano il modello a distinguere le diverse frasi all'interno di un singolo input, categorizzando ogni frase con un parametro diverso.
3. **Position Embedding**: token per indicare la posizione di ogni parola all'interno della sequenza.

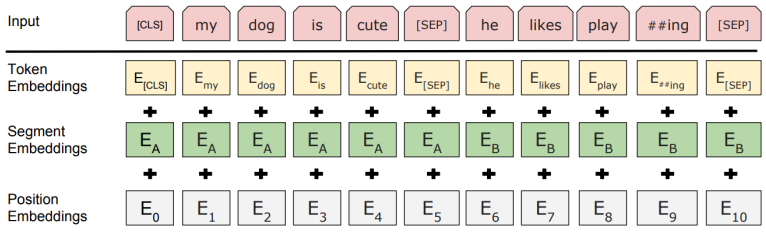

Figura 4.2: Rappresentazione degli input di BERT[6].

Una volta sommati, avremo l'embedding finale che BERT utilizzerà come input.

In seguito la rappresentazione dell'embedding finale passa per i vari strati di Encoder del modello, in cui applica i diversi meccanismi (2.2.1) che arricchiscono il significato delle varie parole.

4.2.1 Masked Language Model(MLM)

Durante la fase di pre-training, il 15% dei token dell'input, scelto casualmente, viene mascherato. L'obiettivo del modello è prevedere quale parola corrisponde a quella originale sostituita con [MASK]. L'input mascherato attraversa i layer dell'Encoder: per ogni livello viene applicato il self-attention per arricchire la rappresentazione di ciascun token.

In seguito passano per i layer di classificazione e genera un vettore di logit. Questo vettore ha all'interno dei valori che rappresentano le parole che il modello ritiene più opportune per la sostituzione dei tag [MASK] e rappresenta l'attention score di ogni token analizzato.

In seguito si applica una funzione softmax, che trasforma i valori del vettore in valori di probabilità.

4.2.2 Next Sentence Prediction(NSP)

Viene preso un set di campioni contenente il 50% di frasi correlate e il 50% di frasi non correlate.
L'obiettivo è capire quali frasi sono collegate tra loro, attraverso l'uscita del valore C che sarà 1 o 0.

Prima di fare ciò, l'input di vettori embedding viene fatto passare dai vari layer dell'Encoder.
In seguito arriva al layer di classificazione che genera, attraverso la funzione di Softmax, la probabilità che una frase sia correlata ad un'altra.

4.2.3 Fine-Tuning

Il fine-tuning è il processo di adattamento di un modello pre-addestrato ad un compito specifico.

1. Preparazione dei Dati
In questa fase avviene la ***tokenizzazione dei dati***, ovvero ogni parola dell'input viene associata a un numero: questo numero rappresenta l'identificativo della parola all'interno del vocabolario di BERT. Inoltre vengono aggiunti i tag [CLS] e [SEP] per identificare l'inizio e la fine di ogni frase, rispettivamente con 102 e 103.

Se prendiamo per esempio la frase

Il modello prepara i dati

sarà suddivisa in

```
['Il', 'modello', 'prepara', 'i', 'dati']
```

In seguito vengono aggiunti i tag di inizio e separazione della frase ed ogni parola viene associata ad un determinato tag. Il vettore è chiamato **tensore.**

```
[[ 102, 329, 3224, 6972, 134, 1484, 103]]
```

2. Definizione del modello Viene definito un modello pre-addestrato in base alle nostre esigenze.
Il modello carica i pesi pre-addestrati che comprendono le conoscenze preacquisite dal pre-training.

3. Addestramento Il modello viene addestrato per alcuni compiti specifici su un set di dati tokenizzato. In questo processo vengono aggiornati i pesi del modello per ridurre gli errori, attraverso una funzione di perdita.

4. Valutazione e Ottimizzazione In questa fase vengono definiti i vari parametri che ci consentono di monitorare la bontà del modello, come **accuratezza, precisione, richiamo e F1-score**.

Particolare attenzione va posta nell'*overfitting*, un fenomeno che si verifica quando un modello è addestrato fin troppo bene, al punto da compromettere la sua capacità di generalizzare nuovi dati mai visti prima: le cause sono relative ai dati di addestramento insufficienti.

4.3 Descrizione del modello pre addestrato

Avendo fatto un'introduzione a BERT, ora analizziamo la scelta del modello pre-addestrato scelto: ***bert-base-uncased***.

Questo modello è stato addestrato con articoli di Wikipedia e da vari testi della raccolta di Corpora OPUS, tramite il MLM e il NSP.

È costituito da:

- **Encoder**: 12
- **Attention head**: 12
- **Dimensione del Feed Forward**: 768
- **Dimensione del vettore di embedding**: 768
- **Token**: *uncased*[3]

4.3.1 Preparazione dei dati

Il fine-tuning per il rilevamento delle PII comporta l'adattamento del modello pre-addestrato ad un **task di classificazione del testo**.
Per far ciò, prima di tutto, dobbiamo effettuare la preparazione dei nostri dati. Essa deve comprendere una struttura chiara e ben definita.

Ogni record di addestramento è composto da una coppia di elementi: **una frase e un tag PII.**
La frase rappresenta il testo che il modello deve analizzare, invece l'etichetta indica la presenza di PII nel testo.

Frase	**PII**
Ho fatto una passeggiata al tramonto e ho scattato belle foto	0
Sono un uomo di 50 anni e la mia email è mariorossi@gmail.com	1

Tabella 4.1: Formattazione dei dati

Durante l'analisi della frase il modello utilizza l'etichetta per determinare se la frase contiene dati sensibili o meno.

La preparazione dei dati costituita in questo modo, consente al modello di apprendere in modo efficace e applicare le conoscenze acquisite durante il processo di fine-tuning. Particolare importanza va data alla qualità e alla precisione di dati per le performance complessive del modello.

[3]Token: uncased indica che il modello non distingue tra lettere maiuscole e minuscole, trattandole allo stesso modo.

4.3.2 Architettura del modello

L'architettura del modello utilizzato è stata descritta precedentemente.
Tuttavia il modello pre-addestrato contiene solo i 12 Encoder che servono a far comprendere il linguaggio, perciò non è addestrato per eseguire un compito specifico.

Per ovviare a ciò, si utilizzano dei layer aggiuntivi all'architettura del modello pre-addestrato BERT, per permettergli di eseguire il compito da noi richiesto. Per questo scopo, utilizziamo **BERTForSequenceClassification**, che estende il modello BERT pre-addestrato per la classificazione del testo, aggiungendo 3 layer:

- **Pooling layer**: Dopo che i token passano attraverso gli Encoder, il layer trasforma la rappresentazione dei token in un'unica rappresentazione che descrive l'intera sequenza.
  L'input che prende in carico è una matrice in cui ogni riga è una rappresentazione arricchita della sequenza iniziale.
  Questo layer si concentra sul token [CLS] (token che viene aggiunto all'inizio di ogni frase) e viene estratta la sua rappresentazione che è considerata una sintesi dell'intera sequenza.
  Il token viene ulteriormente elaborato tramite due strati: ***Linear Layer e Tanh Activation Function***, che hanno lo scopo di aumentare la capacità di BERT nel comprendere le relazioni complesse all'interno della frase.
  L'output quindi sarà l'insieme di tutte le rappresentazioni del token [CLS].

- **Linear Layer**: Questo layer ha il compito di trasformare la rappresentazione dei token in logit, per le classi di output.

- **Output Layer**: I logit passati al layer di Output vengono utilizzati per determinare le classi target.
  Le ***classi target*** sono delle categorie a cui un dato input può appartenere: in questo caso, le nostre classi target saranno **classe 0** (non PII) o **classe 1** (PII).
  I logit prima vengono trasformati in probabilità tramite la funzione *Softmax* nel ***DropOut Layer***.

  In seguito vengono confrontati nel *Prediction Layer*, con l'utilizzo delle etichette, per rappresentare l'appartenenza alle classi target.

Capitolo 5

Implementazione

5.1 Introduzione

In questo capitolo viene illustrata l'implementazione del modello BERT utilizzando il linguaggio di programmazione **Python**.
La scelta di Python è stata motivata dalle tecnologie adottate, in quanto ottimizzate in modo significativo per questo linguaggio, offrendo una maggiore efficienza in termini di prestazioni.
Verrà analizzato il codice e la sua struttura composta da 4 classi: **BertModelTrainier, DataHandler, PIIDataLoader e PIIDataSet**.
Il codice riportato fa particolare attenzione solo ai metodi principali, per visualizzare il codice completo si rimanda alla repository di GitHub

Prima di analizzare in dettaglio la struttura del codice, introduciamo dei concetti fondamentali per capire al meglio il suo funzionamento.

- **Batch**: Un batch rappresenta una suddivisione del dataset. Durante la fase di addestramento i dati vengono elaborati in blocchi(batch) per migliorare l'efficienza computazionale. Ogni batch viene processato in modo indipendente e i suoi dati vengono utilizzati per aggiornare ed ottimizzare i pesi del modello.

- **Epoch**: Un'epoch rappresenta un ciclo completo attraverso l'intero dataset nel processo di addestramento. Durante un'epoch il modello processa ogni esempio nel dataset una volta.
  La scelta del numero di epoch è cruciale per garantire un modello ottimale: è importante sottolineare inoltre un giusto numero di epoch che si aggira tra 3 e 5. Un numero minore fa sì che si verifichi lo stato di **underfitting**, cioè il modello non apprende, o apprende poco, dai dati. Nel caso di un numero elevato, si verifica lo stato di **overfitting**, cioè il modello diventa troppo specializzato, imparando anche il rumore e i dettagli non rilevanti per il suo scopo.

5.2 Tecnologie Utilizzate

Per l'implementazione del modello in Python, sono state impiegate diverse tecnologie.

- **Pytorch**: sviluppata dal laboratorio di ricerca sull'intelligenza artificiale Facebook's AI Research Lab nel 2016, è una libreria open source per il machine learning.
  Sostituisce Numpy per usare e ottimizzare la potenza di calcolo delle schede grafiche (GPU), in quanto Numpy, basando il suo calcolo computazionale sul processore (CPU), allungava i tempi di addestramento.

- **TensorFlow**: sviluppata da Google Brain rilasciata per la prima volta nel 2015, è un framework open-source per il calcolo numerico e l'apprendimento automatico, basati su grafi computazionali.
  Una delle caratteristiche distintive di TensorFlow è l'utilizzo di **tensori**, strutture n-dimensionali, utilizzate per rappresentare i dati nel campo del machine learning.

- **CUDA**: (Compute Unified Device Architecture) è una piattaforma di calcolo parallelo e un modello di programmazione sviluppato da NVIDIA. Introdotta nel 2007 consente agli sviluppatori di utilizzare le GPU per eseguire calcoli su modelli di machine learning.
  È supportato nativamente da Pytorch e TensorFlow.

5.3 Architettura del codice

L'architettura è stata progettata in modo da essere modulare, rendendo semplificata una futura manutenzione del codice.
Il codice è strutturato in tre classi principali ed una ausiliaria: ogni classe è stata scritta in modo da avere un compito specifico, permettendo una chiara separazione dei compiti.

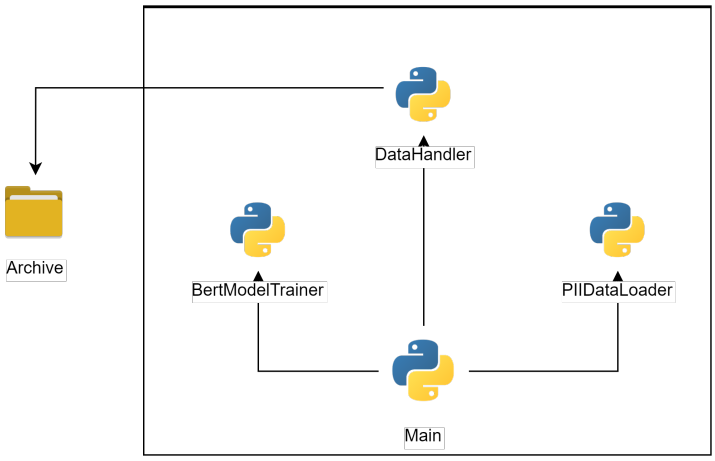

Figura 5.1: Architettura fisica

5.3.1 DataHandler

La classe **DataHandler** è responsabile della gestione e della preparazione dei dati, assicurando che siano strutturati e ben formattati.

Svolge i seguente compiti:

- **Caricamento dei dati**: importa il dataset da un file CSV.
  Questa operazione viene svolta con il metodo *load_data()* che carica i dati dei vari dataset.

- **Pulizia dei dati**: si occupa di eliminare eventuali caratteri prima di essere processati.
  Il metodo *clean_data()* assicura che tutti i dati siano pronti per il successivo utilizzo, riducendo possibili anomalie che potrebbero interrompere l'addestramento del modello.

- **Suddivisione dei dati**: divide il dataset in base al tipo di addestramento che andiamo ad effettuare che potrà essere Training-Test-Validation, K-Fold o Stratified K-Fold grazie ai metodi *split_ data()*, *kfold_ split_ data()* e *stratified_ kfold_ splitdata()*.
  Nel metodo *split_ data()* la dimensione di ogni suddivisione del dataset è già pre impostata quando andiamo ad instanziare un oggetto DataHandler. Nei metodi *kfold_ split_ data()* e *stratified_ kfold_ splitdata()* la suddivisione avviene grazie all'ausilio della libreria Kfold e StratifiedKfold.

```
class DataHandler:
    def __init__(self, dataset_path, train_size=0.8, valid_size=0.1,
    test_size=0.1, n_splits=5):
       [...]

    def load_data(self):
        . . .
    def clean_data(self):
    . . .

    def split_data(self):
        self.df = self.df.sample(frac=1, random_state=123).reset_index(drop
    =True)
        total_size = len(self.df)
        train_end = int(self.train_size * total_size)
        valid_end = train_end + int(self.valid_size * total_size)
        train_texts = self.df.iloc[:train_end]['Sentence'].tolist()
        train_labels = self.df.iloc[:train_end]['Label'].tolist()
        valid_texts = self.df.iloc[train_end:valid_end]['Sentence'].tolist
    ()
        valid_labels = self.df.iloc[train_end:valid_end]['Label'].tolist()
        test_texts = self.df.iloc[valid_end:]['Sentence'].tolist()
        test_labels = self.df.iloc[valid_end:]['Label'].tolist()
        return train_texts, train_labels, valid_texts, valid_labels,
    test_texts, test_labels

    def kfold_split_data(self):
     kf = KFold(n_splits=self.n_splits, shuffle=True, random_state=123)
        sentences = self.df['Sentence'].tolist()
        labels = self.df['Label'].tolist()
        for train_index, test_index in kf.split(sentences):
            train_texts = [sentences[i] for i in train_index]
            train_labels = [labels[i] for i in train_index]
```

```
            test_texts = [sentences[i] for i in test_index]
            test_labels = [labels[i] for i in test_index]
            yield train_texts, train_labels, test_texts, test_labels

    def stratified_kfold_split_data(self):
     skf = StratifiedKFold(n_splits=self.n_splits, shuffle=True,
    random_state=123)
        sentences = self.df['Sentence'].tolist()
        labels = self.df['Label'].tolist()
        for train_index, test_index in skf.split(sentences, labels):
            train_texts = [sentences[i] for i in train_index]
            train_labels = [labels[i] for i in train_index]
            test_texts = [sentences[i] for i in test_index]
            test_labels = [labels[i] for i in test_index]
            yield train_texts, train_labels, test_texts, test_labels

    def get_test_data(self):
         [...]
```

Listing 5.1: Classe DataHandler in Python

5.3.2 PIIDataLoader e PIIDataSet

La classe PIIDataLoader è progettata per gestire il caricamento, la tokenizzazione e suddivisione dei dati in batch.
Si utilizza il tokenizzatore pre-addestrato di BERT, fornito dalla libreria *Hugging Face Transformers* per convertire i testi in input in sequenze di token.
Inoltre la classe definisce i vari metodi per la gestione e la suddivisione dei dati in batch.

- **Inizializzazione dei dati**: Nel suo costruttore vengono definiti i vari parametri, tra cui i testi di training, validazione e testing con le rispettive etichette. Sono pre impostati nulli per permettere di eseguire gli altri tipi di addestramento che non hanno la fase di validazione.
- **Tokenizzazione dei dati:** il metodo *tokenize_ data()* utilizza il tokenizzatore BERT. Come giá accennato nel capitolo 4, la tokenizzazione è un passaggio fondamentale in quanto 'traduce' il testo in una rappresentazione numerica(token).

- **Creazione del dataset**: il metodo *create_ dataset()*, permette la costruzione di dataset Pytorch. Il metodo prende in input le rappresentazioni tokenizzate dei testi (encodings) e le relative etichette (labels) e le unisce in un formato utilizzabile dai DataLoader di Pytorch. In questo modo i dati sono formattati in modo da essere utilizzabili dal modello.

- **Creazione dei DataLoader**: il metodo *get_ dataloader()* combina la tokenizzazione, la creazione del dataset e la suddivisione in batch dei dati. Inoltre con il parametro *shuffle=true* si randomizza la posizione dei dati per esguire il training, una pratica essenziale nell'addestramento di tipo Kfold.

```
class PIIDataLoader:
    def __init__(self, train_texts=None, train_labels=None, valid_texts=
    None, valid_labels=None, test_texts=None, test_labels=None,
    tokenizer_name='google-bert/bert-base-uncased',batch_size=128):
      [...]
    def tokenize_data(self, texts):
        if not texts:
            return None
        return self.tokenizer(texts, truncation=True, padding=True,
    return_tensors="pt")

    def create_dataset(self, encodings, labels):
        return torch.utils.data.Dataset(encodings, labels)
    def get_specific_dataloader(self, mode):
      [...]

    def get_dataloader(self):
        train_encodings = self.tokenize_data(self.train_texts)
        valid_encodings = self.tokenize_data(self.valid_texts)
        test_encodings = self.tokenize_data(self.test_texts)
        train_dataset = PIIDataset(train_encodings, self.train_labels)
        valid_dataset = PIIDataset(valid_encodings, self.valid_labels)
        test_dataset = PIIDataset(test_encodings, self.test_labels)
        train_loader = torch.utils.data.DataLoader(train_dataset,
    batch_size=self.batch_size, shuffle=True)
        valid_loader = torch.utils.data.DataLoader(valid_dataset,
    batch_size=self.batch_size, shuffle=False)
        test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=
    self.batch_size, shuffle=False)
        return train_loader, valid_loader, test_loader
```

Listing 5.2: Classe PIIDataLoader in Python

Inoltre si occupa della creazione di dataset Pytorch personalizzati, attraverso la classe ausiliare PIIDataset, che carica i dati tokenizzati e li associa alle etichette corrispondenti.

```
class PIIDataSet:


    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels


    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.
    items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item









    def __len__(self):
        return len(self.labels)
```

Listing 5.3: Classe PIIDataSet in Python

5.3.3 BertModelTrainer

La classe BertModelTrainer si occupa dell'addestramento, valutazione,testing e gestione del modello BERT.
Il modello pre-addestrato di BERT è fornito dalla libreria Hugging Face Transformer.
Inoltre viene istanziato ***BertForSequenceClassification***, un'aggiunta di 3 layer finali. Grazie ad esso è possibile effettuare la classificazione del testo.

- *__init()__*: Il metodo init è il costruttore della classe e si occupa dell'inizializzazione di tutte le variabili necessarie per il processo di addestramento.

```
def __init__(self, model_name, device, num_epochs=3, lr=5e-5):
    self.model_name = model_name
    self.device = device
    self.num_epochs = num_epochs
    self.lr = lr
    self.model = BertForSequenceClassification.from_pretrained(
model_name)
    self.model.to(device)
    self.optim = torch.optim.Adam(self.model.parameters(), lr=lr)
    self.train_losses = []
    self.valid_accuracies = []
```

Listing 5.4: Metodo __init()__

- *train()*: si occupa di eseguire l'addestramento del modello. Il metodo inoltre tiene conto della perdita (loss) di ciascun batch[1] e nel caso fosse disponibile un set di validazione, valida il modello durante il processo di addestramento.

```
  def train(self, train_loader, valid_loader=None):
    for epoch in range(self.num_epochs):
        self.model.train()
        batch_losses = []
        for batch_idx, batch in enumerate(train_loader):
            input_ids = batch['input_ids'].to(self.device)
            attention_mask = batch['attention_mask'].to(self.
device)
            labels = batch['labels'].to(self.device)
            outputs = self.model(input_ids, attention_mask=
attention_mask, labels=labels)
            loss = outputs.loss
            self.optim.zero_grad()
            loss.backward()
            self.optim.step()
            batch_losses.append(loss.item())
            [...]
        # Calculate average loss for the epoch
        epoch_loss = sum(batch_losses) / len(batch_losses)
```

[1]La perdita di ciascun batch si riferisce al processo di calcolo e memorizzazione del loss(perdita), in modo da aggiornare ogni gradiente alle fine di un epoch. .

```
            self.train_losses.append(epoch_loss)
            # If valid_loader is provided, compute validation accuracy
            if valid_loader is not None:
                valid_accuracy = self.compute_accuracy(valid_loader)
                self.valid_accuracies.append(valid_accuracy)
                [...]
            else:
               [...]
        [...]
```

Listing 5.5: Metodo train()

Il **reset** dei gradienti attraverso *self.optim.zero_grad()* permette di evitare di sommare il gradiente al batch precedente. Questa operazione è fondamentale in quanto in Pytorch gli ottimizzatori accumulano i gradienti.

Viene eseguita la **backpropagation** tramite *loss.backward()* calcolando il gradiente della perdita rispetto ai pesi del modelli.

Infine si utilizza *self.optim.step()* per utilizzare i gradienti per **aggiornare i pesi** del modello, un'azione necessaria per migliorare la performance del modello.

- *evaluate()*: misura l'accuratezza del modello tramite il testing su un set di dati.
- *compute_accuracy()*: calcola la precisione sui set di validazione o di test.
- *save_model()*: consente di salvare lo stato del modello.

```
def save_model(self, model_path, optimizer_path):
#salva lo stato del modello attraverso i tensori dei pesi
     torch.save(self.model.state_dict(), model_path)
 #salva lo stato dellottimizzazione del modello, tramite i
   parametri appresi durante l'addestramento
      torch.save(self.optim.state_dict(), optimizer_path)
```

Listing 5.6: Metodo save_model()

- *load_model()*: consente di caricare lo stato del modello.
- *plot_metrics()*: genera grafici delle metriche di addestramento e validazione.

5.4 Esempio di implementazione

In questa sezione viene illustrata l'implementazione del modello. Il codice descrive il processo del caricamento del dataset e della preparazione dei dati tramite DataHandler e PIIDataLoader, ed infine il training e la valutazione del modello tramite BertModelTrainer.

In questo esempio verrà implementato un addestramento tramite Training Validation Test, una tecnica (si introdurrà nel capitolo 6) in cui i dati vengono suddivisi in: 80% dati di training, 10% dati di testing e il restante 10% in dati di valutazione.

```
#Vengono utilizzati algoritmi deterministici. Ogni volta che si addestra
 il modello i risultati saranno gli stessi.
 torch.backends.cudnn.deterministic = True
 RANDOM_SEED = 123
 torch.manual_seed(RANDOM_SEED)
 DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
 NUM_EPOCHS = NUM_EPOCH
 # Carica e processa i dati
 dataset_path = [...]
 data_handler = DataHandler(dataset_path)
 data_handler.load_data()
 data_handler.clean_data()
 train_texts, train_labels, valid_texts, valid_labels, test_texts,
 test_labels = data_handler.split_data()
 # Crea data loader per la tokenizzazione dei dati
 data_loader = PIIDataLoader(train_texts, train_labels, valid_texts,
 valid_labels, test_texts, test_labels)
 train_loader, valid_loader, test_loader = data_loader.get_dataloader()
 # Addestra e valuta il modello
 model_trainer = BertModelTrainer(model_name='google-bert/bert-base-
 uncased', device=DEVICE, num_epochs=NUM_EPOCHS)
 model_trainer.train(train_loader, valid_loader)
 model_trainer.evaluate(test_loader)
 # Salvataggio del modello
 model_trainer.save_model('bert_model.pt', 'optimizer.pt')
 torch.save(model_trainer,path)
 model_trainer.plot_metrics()
    torch.save(self.optim.state_dict(), optimizer_path)
```

Listing 5.7: Esempio di implementazione

1. **Utilizzo di algoritmi deterministici e impostazioni del dispositivo**
   La scelta di utilizzo di algoritmi deterministici è motivata dalla necessità di comprendere a pieno il suo funzionamento e di garantire riproducibilità dei risultati. Nonostante i vari test eseguiti abbiano prodotto dei risultati ottimali, lo scopo principale è l'ottimizzazione del modello anche per scenari futuri. Con un dataset ancora più ricco ed ottimizzato, sarà possibile l'utilizzo di algoritmi non deterministici. La scelta del dispositivo di calcolo è stata motivata dall'efficienza computazionale: utilizzando PyTorch, l'implementazione di CUDA è una scelta essenziale per ottimizzare i tempi di addestramento.

2. **Scelta del dataset e suddivisione dei dati**
   Viene istanziato un oggetto DataHandler in modo che la classe pulisca, carichi e suddivida i vari dati in base al tipo di training che andremo ad effettuare. Questa suddivisione è fondamentale per garantire una corretta valutazione del modello.

3. **Tokenizzazione dei dati**
   Come discusso precedentemente, la classe PIIDataLoader si occupa della tokenizzazione dei dati, grazie al tokenizzatore fornito da Hugging Face. Questa fase è cruciale per garantire la conversione dei dati in un formato compatibile al modello.

4. **Addestramento e valutazione del modello**
   In questa sezione viene specificato il modello pre-addestrato. Inoltre vengono suddivisi i dati in batch per facilitare l'addestramento e ottimizzare l'utilizzo della memoria.

5. **Salvataggio del modello**
   Al termine dell'addestramento lo stato del modello e dell'ottimizzatore vengono salvati.

Capitolo 6

Validazione

6.1 Introduzione

In questo capitolo si descrivono i diversi approcci di addestramento impiegati per l'addestramento del modello BERT, con un'analisi dettaglia delle metodologie utilizzate, quali **Training-Validation-Test, K-Fold e Stratified K-Fold.**

In seguito si discutono le **metriche per la valutazione**, andando a valutare ogni modello per tipo di addestramento. Inoltre saranno descritti i vari dataset per l'addestramento e il testing, cercati dopo un'attenta ricerca per garantire la bontà dei dati, e quindi per evitare situazioni di overfitting o underfitting.

Prima di sviluppare il capitolo si introducono le seguenti definizioni.

- **Dati di addestramento**: dati utilizzati per far apprendere al modello il suo "comportamento". Sono i dati che cambiano le metriche e i gradienti andando a migliorare l'efficienza del modello.
  Questo tipo di apprendimento permette di ridurre la funzione di perdita, migliorando la capacità del modello nel fare previsioni accurate.
  Questo processo, noto come **backpropagation**, consente al modello di comprendere i pattern sottostanti, ottimizzando la generalizzazione in caso di dati futuri non ancora visti[7].

- **Dati di validazione**: dati che rappresentano un sottoinsieme del dataset utilizzato per valutare le prestazioni durante l'addestramento. Non influenzano direttamente il processo di aggiornamenti dei pesi, ma aiutano a fornire informazioni per orientare le scelte di ottimizzazione del modello.
- **Dati di testing**: dati utilizzati per misurare le prestazioni di un modello dopo che l'addestramento è stato completato. Diversamente dai dati di validazione, questo set di dati non è utilizzato per influenzare i pesi, ma fornisce una valutazione delle capacità del modello di generalizzare su dati nuovi.

6.1.1 Sviluppo del lavoro

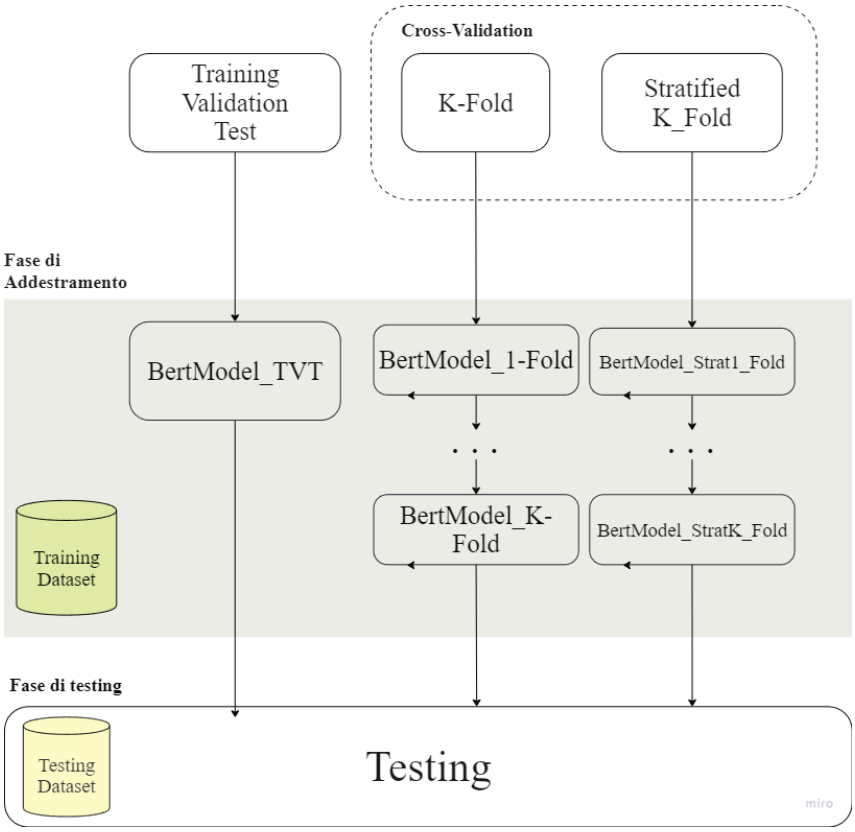

Figura 6.1: Flusso di lavoro

Come illustrato dalla figura lo sviluppo del lavoro si articola in due distinte fasi: **fase di addestramento e fase di testing**.

Durante la fase di addestramento, dopo aver selezionato le tecniche da utilizzare, il modello verrà addestrato sul dataset di training.
Otterremo 3 tipologie di modelli:

- **BertModel_TVT:** Questo modello si ottiene tramite l'approccio Training-Validation-Test. Ciò implica una suddivisione classica del dataset in tre parti distinte.
- **BertModel_K-Fold:** Questo modello si ottiene tramite l'approccio K-Fold Cross-Validation, risultando nella creazione di K modelli distinti, in base alla suddivisione di Fold effettuate.
- **BertModel_StratK-Fold** : Similmente alla tecnica K-Fold, tale approccio assicura che ogni fold contenga una rappresentazione proporzionale di ciascuna classe di output, garantendo una distribuzione più omogenea dei dati.

Nella **fase di testing** verranno testati i vari modelli tramite le metriche **Accuracy,Precision, Recall e F1-Score**.

6.2 Metodologie di addestramento

6.2.1 Training-test-Validation

Nella fase di addestramento di un modello di machine learning, un parte cruciale consiste nel suddividere i dati per **addestrare, testare e validare**.
Tale suddivisione ci permette di sviluppare un modello che non solo apprenda dai dati, ma che possieda anche la capacità di generalizzare su nuovi set di dati precedentemente non analizzati.

Come illustrato nella figura i dati si suddividono in:

- 80% dati di addestramento
- 10% dati di valutazione
- 10% dati di testing

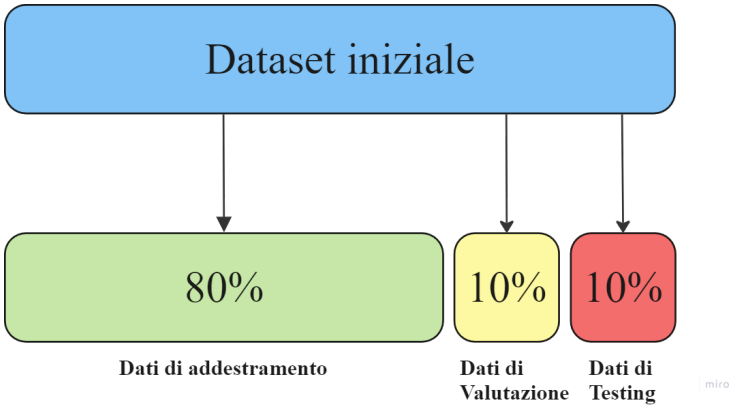

Figura 6.2: Suddivisione dei dati

L'impiego di questa suddivisione è giustificata da una fase di testing con un dataset estraneo da quello iniziale.
Ciò ci permette di garantire una valutazione più robusta e oggettiva della capacità del modello per il rilevamento delle PII.
L'adozione di questa tecnica di suddivisione del dataset consente di implementare un modello che ha una fase di validazione, elemento che nelle altre tecniche utilizzate non è presente.

L'obiettivo di questa strutturazione è di effettuare una compararazione con gli altri modelli ottenuti tramite una diversa suddivisione del dataset. Questo approccio ci permette di analizzare le loro metriche e la loro efficacia nel rilevamento delle PII.

6.2.2 Cross Validation

La tecnica **Cross-Validation** è uno dei concetti piú importanti nel machine learning. Consiste nella suddivisione dei nostri dati di addestramento in due diverse porzioni: la prima relativa ai dati di addestramento

e la seconda relativa ai dati di testing. Questo approccio ci permette di "generalizzare" il comportamento del modello, in modo che il modello riesca a eseguire delle decisioni anche su dati non ancora visti.

Tale tecnica consiste nello specifico nella suddivisione del dataset in **fold**, sottoinsiemi utilizzati per condurre addestramenti e testing.
Il principio fondamentale è che un fold viene utilizzato,ad ogni iterazione, per effettuare il testing sul modello dopo che esso viene addestrato sulla restante parte del dataset.

Esistono diverse varianti di Cross Validation, ciascuna adatta a specifiche necessità. Nel contesto di questa tesi si adottano due metodologie: **K-Fold e Stratified K-Fold**.

K-Fold

Nella metodologia **K-Fold Cross Validation**, il dataset viene suddiviso in 2 parti: **training set e testing set**.
In genere, il training set rappresenta l'80% del dataset, e il restante 20% è usato per il testing.
La caratteristica peculiare del K-Fold è la suddivisione del dataset in K fold diverse e la ripetizione del processo di addestramento per **K iterazioni**.

Durante ciascuna iterazione,il fold viene utilizzato per il testing, mentre le altre **K-1** parti vengono utilizzate per il training. In questo modo il modello viene testato su tutte le parti del dataset almeno una volta, riducendo il rischio di overfitting.

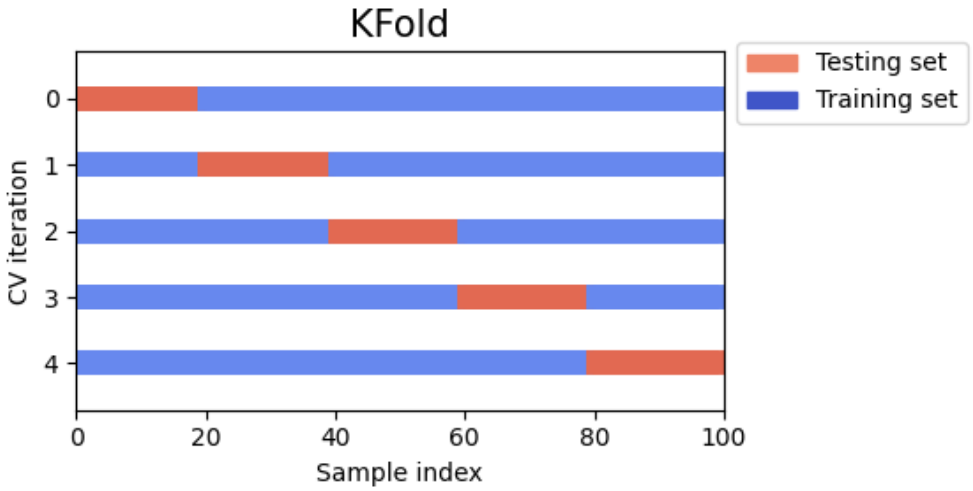

Figura 6.3: Suddivisione dei dati K-Fold

Stratified K-Fold

Nella metodologia **Stratified K-Fold Cross validation**, il dataset viene suddiviso in due parti principali: training set e testing set.
Generalmente, come il K-Fold, il training set rappresenta l'80% del dataset totale, mentre il restante 20% è destinato al testing.

La caratteristica che lo distingue dal K-Fold è il modo in cui i vari dati vengono suddivisi. Si mantiene la stessa **distribuzione delle classi**: ció significa che all'interno di un fold ci saranno 50% dati contententi PII e 50% di dati non contenenti PII.
Questa tecnica è particolarmente utile quando il dataset è sbilanciato e ci permette di effettuare il testing su un set di dati bilanciato tra le varie classi.

Come nel K-Fold, il dataset viene suddiviso in K fold e il processo di addestramento viene ripetuto K volte.Durante ciascuna iterazione,il fold viene utilizzato per il testing, mentre le altre K-1 parti vengono utilizzate per il training.

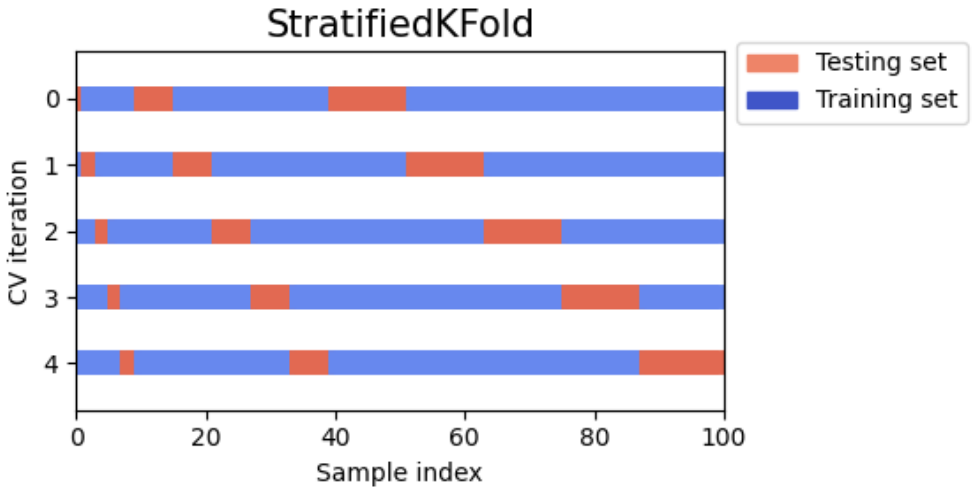

Figura 6.4: Suddivisione dei dati Stratified K-Fold

6.3 Training Dataset

La scelta di un dataset di addestramento è fondamentale per lo sviluppo di un modello di machine learning.
La lingua scelta per il dataset è l'**inglese**, al fine di garantire una più ampia applicabilità del modello.

Dopo un'accurata indagine sui dataset disponibili che includono un'ampia varietà di categorie PII, è stato selezionato il dataset ***pii-masking-200k[8]***.
Questo dataset si distingue per la sua varietà di dati, contenendo 54 classi di PII distribuite su 229 casi d'uso e discussioni in diversi ambienti quali il business,l'educazione, la psicologia e il settore legale.
Inoltre include 5 diverse tipologie di interazione tra cui conversazioni formali, casuali, email ecc..
Tale dataset comprende diversi dati in in 4 lingue diverse tra cui inglese, francese, tedesco e italiano.
Visto la scelta della lingua sono stati selezionati tutti i campi in inglese, identificati dalla colonna language con il valore 'eng', portando all'ottenimento di circa **43.000 righe**, che costituiranno la base per l'addestramento del modello.

Le **classi PII** presenti nel dataset sono:

VLITECOINADDRESS	VEHICLEVIN
DATE	JOBTYPE
GENDER	ORDINALDIRECTION
BIC	MASKEDNUMBER
URL	STATE
IP	IPV4
STREET	CURRENCYCODE
HEIGHT	CURRENCYSYMBOL
LASTNAME	BUILDINGNUMBER
VEHICLEVRM	NEARBYGPSCOORDINATE
SECONDARYADDRESS	JOBTITLE
IPV6	IBAN
SEX	COMPANYNAME
PIN	CREDITCARDCVV
DOB	PREFIX
AMOUNT	EYECOLOR
TIME	ACCOUNTNUMBER
CREDITCARDISSUER	COUNTY
ETHEREUMADDRESS	ACCOUNTNAME
MIDDLENAME	USERAGENT
BITCOINADDRESS	PHONENUMBER
ZIPCODE	AGE
MAC	PASSWORD
PHONEIMEI	
USERNAME	
CREDITCARDNUMBER	

Tabella 6.1: Classi PII presenti nel dataset

La rilevanza di questo dataset è stata ampiamente riconosciuta dai vari articoli scientifici, come lo dimostrano le citazioni nei paper [9][10][11][12]. In particolare, nel [9] si evidenzia l'importanza del dataset per promuovere lo sviluppo di sistemi automatici per la de-identificazione, mentre nel paper [11] si evidenzia il contesto sperimentale realistico offerto.

6.3.1 Preparazione dei dati

Il dataset scelto, è stato progettato per modelli di tipo Named Entity Recogniticon(NER) e contiene esclusivamente dati con identificatori personali(PII).

Per tali modelli, la scelta di questo singolo dataset può essere sufficiente, in quanto ogni parola discriminante è annotata con un tag che ne identifica la natura PII, permettendo un'analisi dettagliata di tutte le parole.

Nonostante la bontà dei dati di tale dataset, è emersa la necessità di integrare un dataset aggiuntivo che non includa PII per bilanciare l'addestramento e garantire la sua efficacia in contesti più ampi.
A tale scopo è stato preso il dataset ***Generic-Sentiment***[13],disponibile su **Kaggle**, piattaforma che mette a disposizione dataset di pubblico dominio.

Generic-Sentiment comprende frasi generiche raccolte da diversi contesti, il che lo rende ideale per un addestramento equilibrato del modello.
Le righe selezionate sono state 43.000 prese randomicamente sulle 50.000 messe a disposizione in modo che, la percentuale tra dati contenenti PII e non, sia circa il 50%[1].

Figura 6.5: Suddivisone Training Dataset

[1]Si parla di circa il 50% in quanto all'interno del dataset sono stati trovati alcuni dati corrotti, che sono stati rimossi.

6.4 Fase di Training

Nella fase di testing, analizziamo i vari modelli sviluppati e le loro rispettive metriche.

I parametri utilizzati durante la fase di training sono:

- **Numero di Epoch**: 3
- **Dimensione del Batch**: 64
- **Numero di fold**: 5

Prima di introdurre i risultati, definiamo una leggenda per la comprensione dei dati.

- **Training Loss**: rappresenta l'errore del modello calcolato sui dati di addestramento. Si verifica durante il training e quantifica le previsioni non corrette che il modello ha effettuato.
- **Validation accuracy**: rappresenta la percentuale di campioni correttamente classificati dal modello sul **set di validazione**.
- **Test accuracy**: l'accuracy misura la percentuale di previsioni corrette rispetto alla previsioni effettuate(6.6.1).

Score Training-Validation-Test

Epoch	Training Loss	Validation Accuracy
1	1,66%	99,93%
2	0,26%	99,92%
3	0,20%	99,85%

Total Training Time	Test Accuracy
32,22 min	**99,86129%**

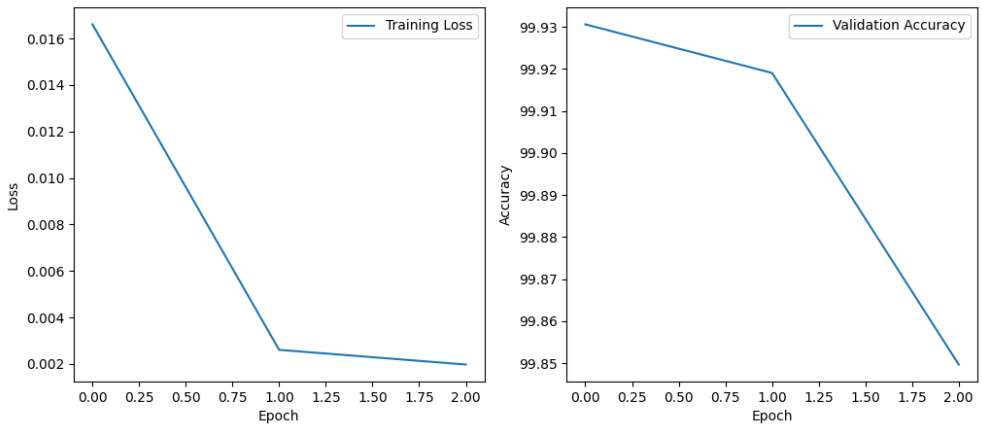

Figura 6.6: Grafico dell'andamento dei parametri durante le varie epoch

Score K-Fold 1

Epoch	Training Loss
1	1.44%
2	0.28%
3	0.20%

Total Training Time	Test Accuracy
31,39 min	**99,9108%**

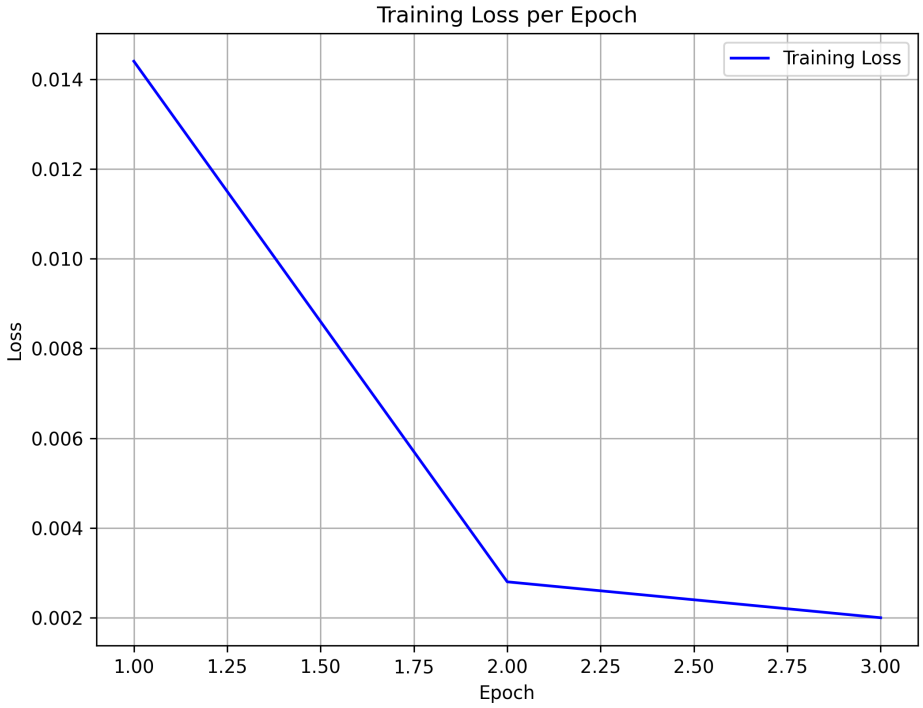

Figura 6.7: Grafico dell'andamento dei parametri durante le varie epoch per Kfold 1

Score K-Fold 2

Epoch	Training Loss
1	1.42%
2	0.25%
3	0.19%

Total Training Time	Test Accuracy
31.18 min	**99.85549%**

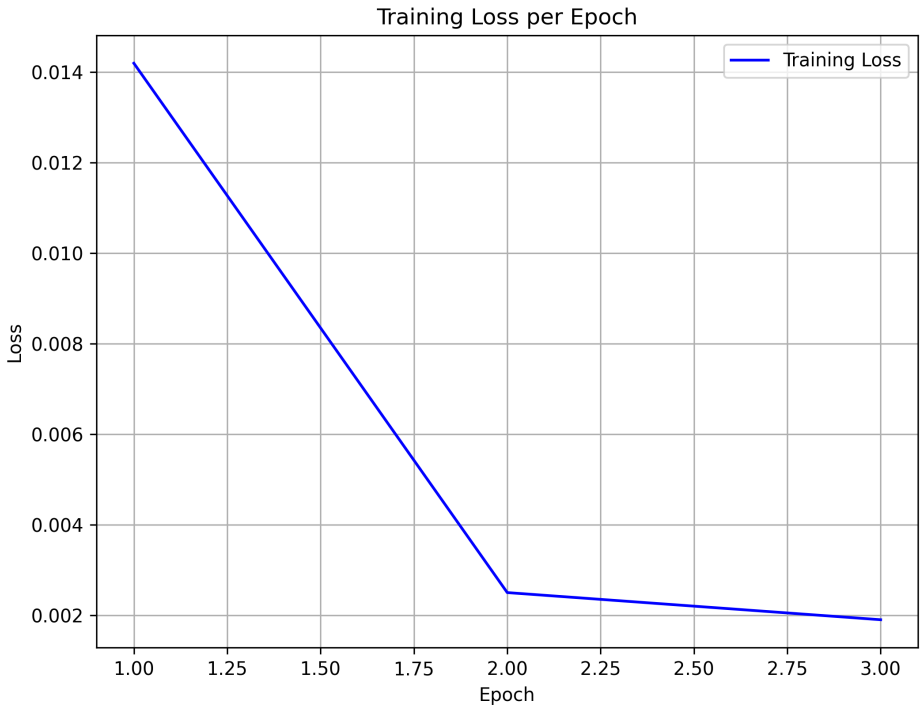

Figura 6.8: Grafico dell'andamento dei parametri durante le varie epoch per Kfold 2

Score K-Fold 3

Epoch	Training Loss
1	1,45%
2	0.27%
3	0.25%

Total Training Time	Test Accuracy
30.55 min	**98.90751%**

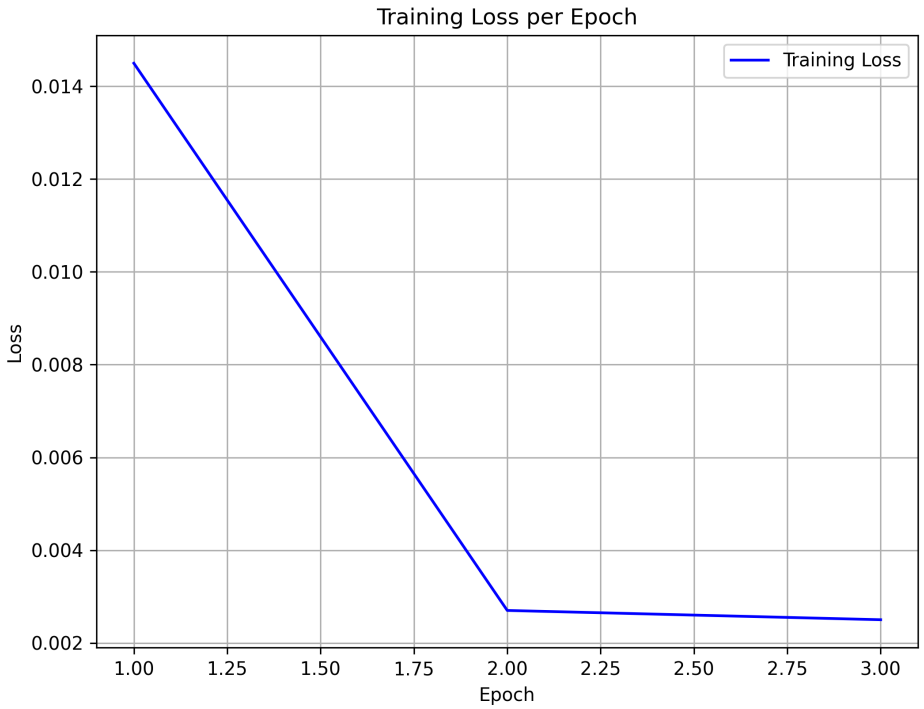

Figura 6.9: Grafico dell'andamento dei parametri durante le varie epoch per Kfold 3

Score K-Fold 4

Epoch	Training Loss
1	1,43%
2	0.30%
3	0.31%

Total Training Time	Test Accuracy
31.15 min	**99.84393%**

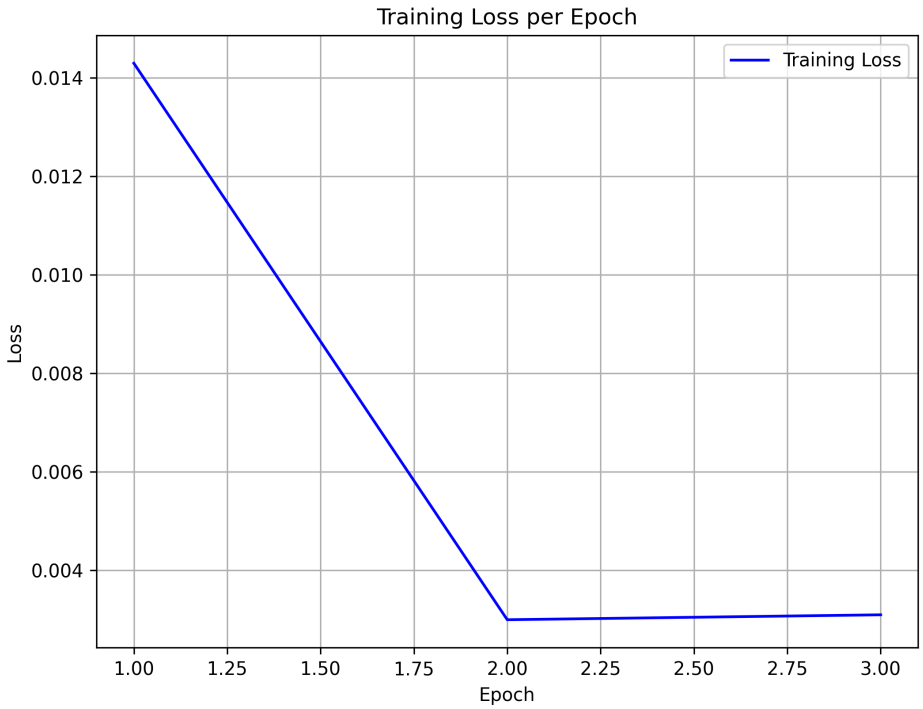

Figura 6.10: Grafico dell'andamento dei parametri durante le varie epoch per Kfold 4

Score K-Fold 5

Epoch	Training Loss
1	1,33%
2	0.28%
3	0.23%

Total Training Time	Test Accuracy
31.15 min	**99.84393%**

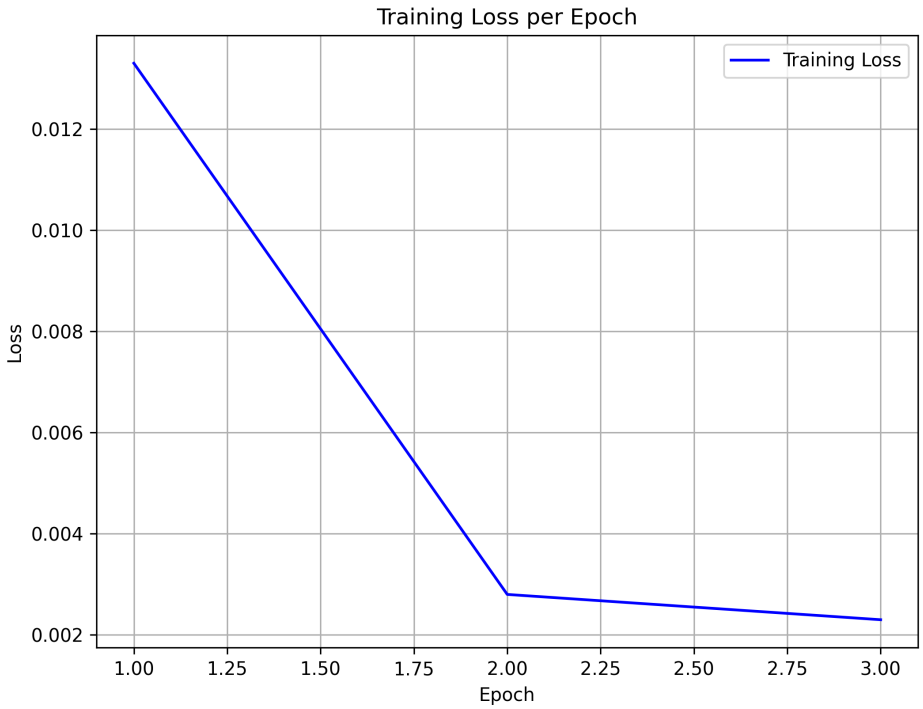

Figura 6.11: Grafico dell'andamento dei parametri durante le varie epoch per Kfold 5

Score Stratified K-Fold 1

Epoch	Training Loss
1	1,62%
2	0.21%
3	0.22%

Total Training Time	Test Accuracy
30.59 min	**99.61850**%

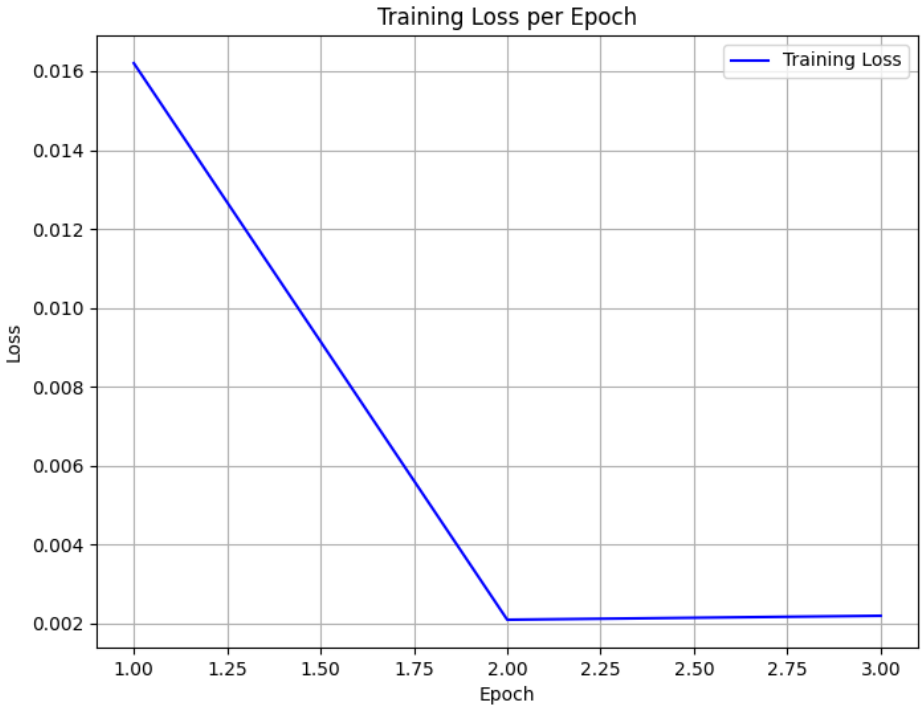

Figura 6.12: Grafico dell'andamento dei parametri durante le varie epoch per Stratified K-Fold 1

Score Stratified K-Fold 2

Epoch	Training Loss
1	1,29%
2	0.32%
3	0.17%

Total Training Time	Test Accuracy
31.03 min	**99.93064%**

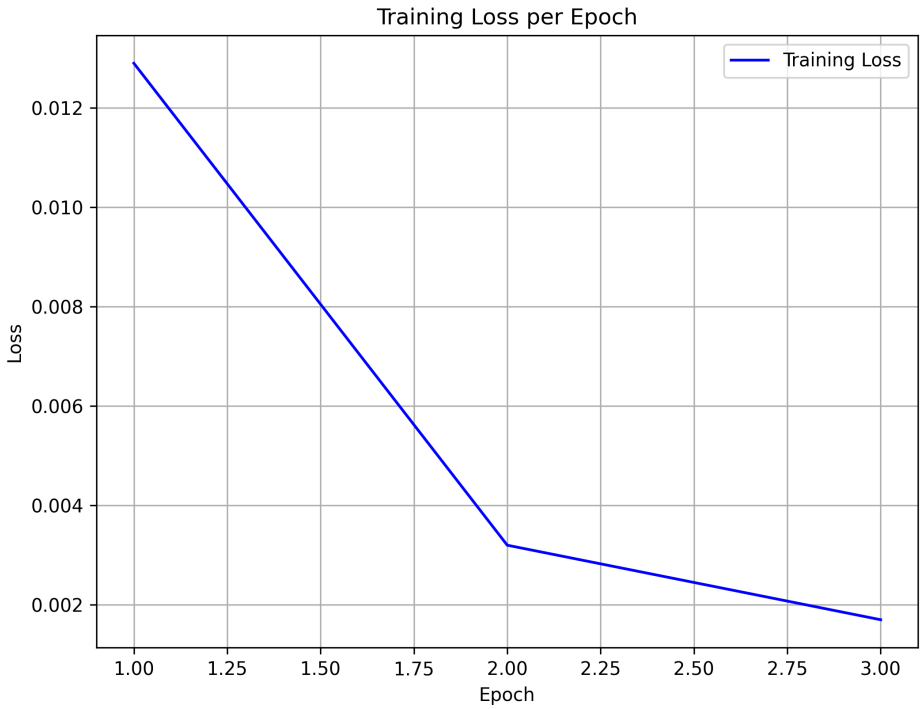

Figura 6.13: Grafico dell'andamento dei parametri durante le varie epoch per Stratified K-Fold 2

Score Stratified K-Fold 3

Epoch	Training Loss
1	1,50%
2	0.20%
3	0.29%

Total Training Time	Test Accuracy
30.54 min	**99.78035%**

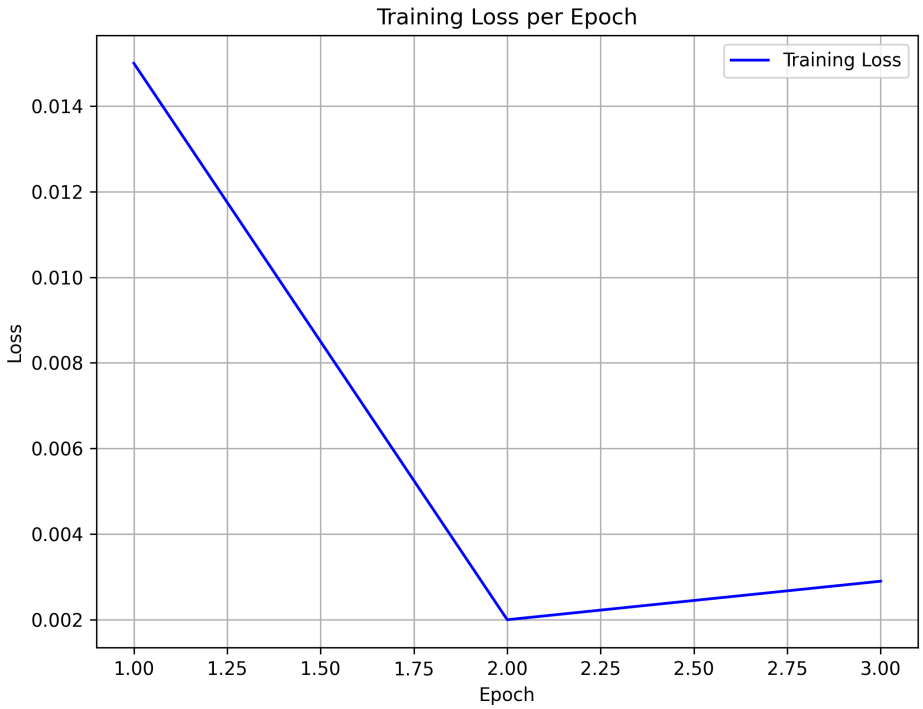

Figura 6.14: Grafico dell'andamento dei parametri durante le varie epoch per Stratified K-Fold 3

Score Stratified K-Fold 4

Epoch	Training Loss
1	1,42%
2	0.28%
3	0.22%

Total Training Time	Test Accuracy
31.00 min	**99.83815%**

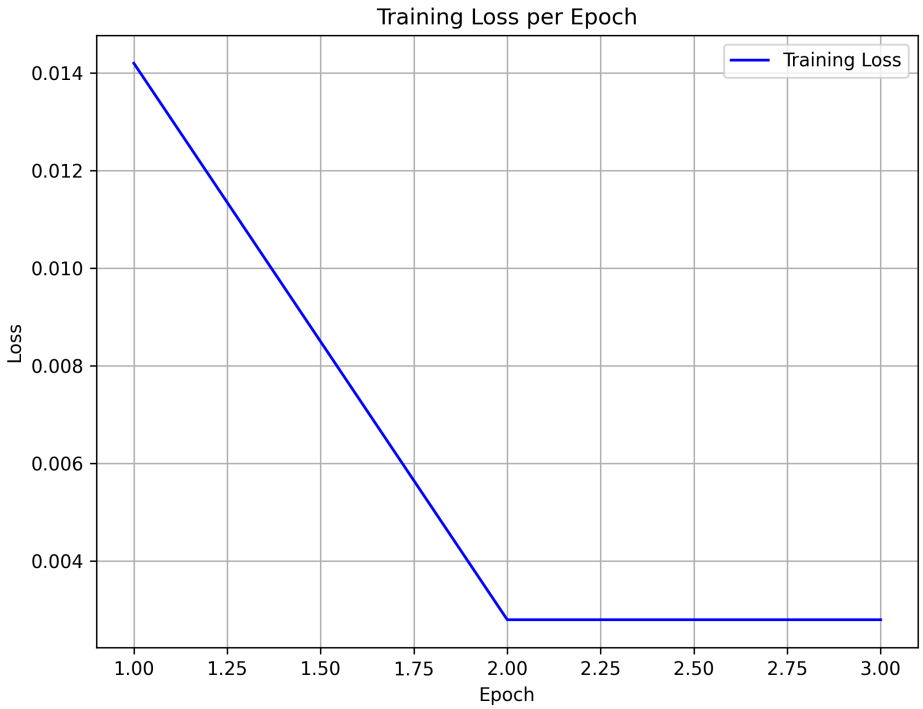

Figura 6.15: Grafico dell'andamento dei parametri durante le varie epoch per Stratified K-Fold 4

Score Stratified K-Fold 5

Epoch	Training Loss
1	1,45%
2	0.21%
3	0.25%

Total Training Time	Test Accuracy
30.52 min	**99.91329%**

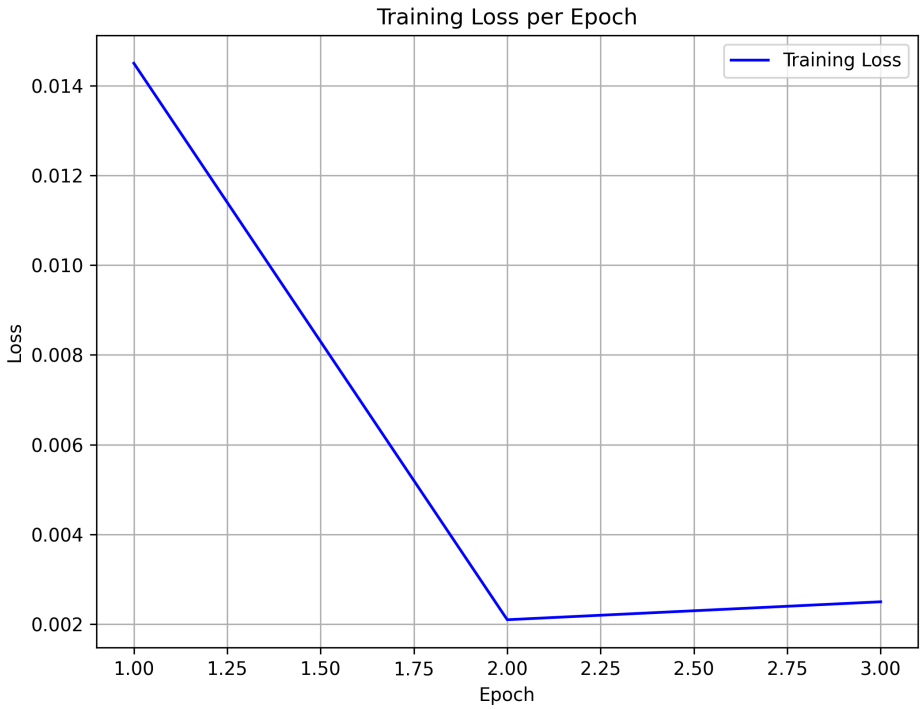

Figura 6.16: Grafico dell'andamento dei parametri durante le varie epoch per Stratified K-Fold 5

6.4.1 Analisi dei dati

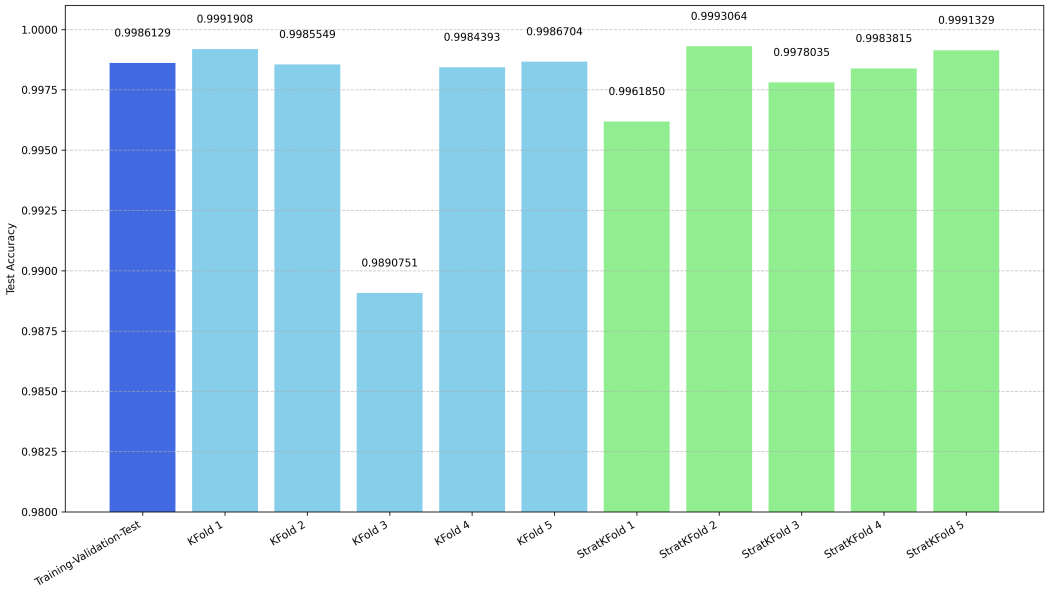

Figura 6.17: Confronto tra i modelli

Dall'analisi dei dati risulta emergere che la variazione massima tra un modello e l'altro è del 1,02313%, con un media del test accuracy pari a 99.6786% per i modelli addestrati con la tecnica **K Fold** e del 99,81619% per i modelli addestrati tramite la tecnica **Stratified K Fold**.
Sommariamente la tecnica con più affidabilità è lo Stratified KFold.
Il miglior modello è *Stratified KFold* 2 con un test accuracy del 99.93064%.

Sebbene la differenza sia di pochi punti percentuali, nelle fasi successive analizzeremo il comportamento di tutti i modelli utilizzando un dataset di testing esterno.

6.5 Testing dataset

Dopo aver osservato le performance dei modelli dopo la fase di training, resta la scelta di composizione di dataset esterno per il testing.
La sua struttura si compone da vari dataset riguardanti diverse categorie.

- **pii-masking-43k**[14]: rilasciato da Ai4Privacy è stato selezionato per l'affidabilità dei dati prodotti da esso. Tale dataset in particolare, è stato impiegato dal gruppo per il fine-tuning di Distilled BERT, ottenendo risultati iniziali di elevata precisione (99,8636%), recall(99,8945%) e accuratezza(99,4532%). Dopo un'attenta valutazione, pii-masking-43k è stato utilizzato per integrare il dataset di testing, garantendo la diversità della tipologia dei dati rispetto a pii-masking-200k.

- **Dialog Dataset**[15]: Contenente conversazioni prive di PII, è stato scelto per diversificare i dati di testing. È specificatamente orientato a scenari di dialogi di chatbot e per il testing sono state selezionate ***1.000 righe***.

- **Movie Review** [16]: *"IMDB Dataset of 50K Movie Reviews"* . è considerato un 'pilastro' per i modelli NLP, essendo già predisposto per l'addestramento di classificazione del testo. Utilizzato principalmente per l'analisi del sentiment, le varie recensioni sono etichettate 'positive' e 'neutral'. Accertato che non contenesse PII, è stato adeguatamente modificato ed integrato nel dataset di testing, selezionando ***10.000 righe*** .

- **Chat-GPT 4 Sentences**: È stato chiesto al modello GPT-4 la generazione di ***1000 righe*** che non contenessero PII, per testare la capacità del modello di gestire dati generati artificialmente.

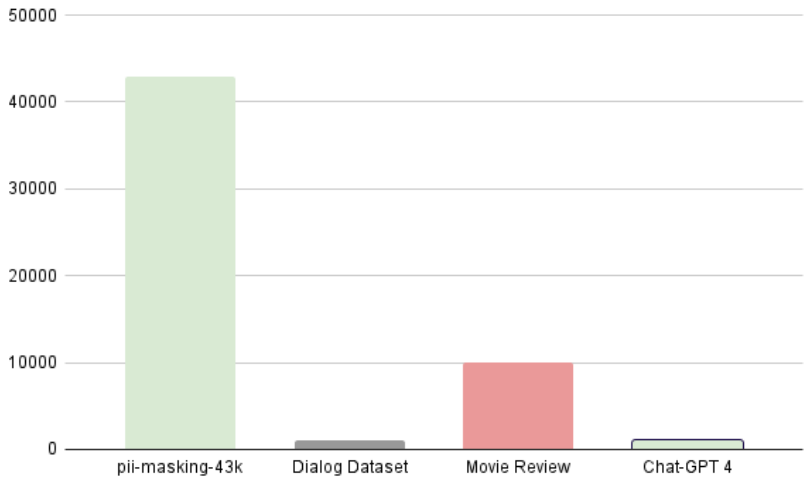

Figura 6.18: Suddivisone Training Dataset

6.6 Fase di Testing

6.6.1 Metriche di valutazione

Dopo aver definito il nostro dataset di testing, procediamo alla **valutazione dei modelli** presi in considerazione.

Nel campo del machine learning, è necessario utilizzare diverse metriche per valutare correttamente le prestazioni dei vari modelli.
Esse forniscono **insight critici** sulla capacità del modello di fare previsioni accurate e su come esso gestisce le classi sbilanciate.
Indicheremo con:

- **TP**: true positive(vero positivo)
- **TN**: true negative (vero negativo)
- **FP**: false positive(falso positivo)
- **FN**: falso negativo (falso negativo)

Accuracy (accuratezza)

L'**accuracy** misura la percentuale di previsioni corrette rispetto alla previsioni effettuate.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision (precisione)

La **precisione** rappresenta le identificazioni positive che sono effettivamente corrette rispetto ai falsi positivi.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (sensibilità)

La **sensibilità** rappresenta la percentuale dei veri positivi che sono stati identificati correttamente rispetto ai falsi negativi.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-Score

L'**F1-Score** è la media armonica di precisione e recall fornendo uno score che bilancia entrambe le metriche, particolarmente utile per le classi sbilanciate

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

6.6.2 Score

	Test Accuracy	Precision	Recall	F1-score
Training-Test-Validation	99,55%	99,56%	99,55%	99,55%
K-Fold				
K Fold 1	99,24%	99,26%	99,24%	99,24%
K Fold 2	98,90%	98,95%	98,90%	98,91%
K Fold 3	80,84%	89,39%	80,84%	82,21%
K Fold 4	97,57%	98,10%	97,57%	97,59%
K Fold 5	99,34%	99,36%	99,34%	99,34%
Stratified K-Fold				
Str. K Fold 1	98,48%	98,49%	98,48%	98,48%
Str. K Fold 2	98,65%	98,72%	98,65%	98,66%
Str. K Fold 3	95,95%	96,54%	95,95%	95,95%
Str. K Fold 4	99,45%	99,45%	99,45%	99,45%
Str. K Fold 5	99,41%	99,43%	99,41%	99,41%

Tabella 6.2: Risultati ottenuti

Dopo aver testato i vari modelli e analizzate le loro prestazioni tramite le metriche **Accuracy,Precision, Recall, e F1 Score**, passiamo ad analizzare i risultati ottenuti.

Il miglior modello, sebbene con un piccolo margine di percentuale, è risultato essere, su tutti i campi, quello addestrato con la tecnica del Training-Validation-Test.
I risultati ottenuti per questo modello sono stati i seguenti:

- **Accuracy**: 99.55823%
- **Precision**: 99.564%
- **Recall**: 99.558%
- **F1 Score**: 99.559%

In contrapposizione con le aspettative basate sulla fase di training, dove il modello *Stratified KFold 2* aveva mostrato la migliore Test Accuracy sui dati iniziali.
Questa discrepanza evidenzia **l'importanza della fase di validazione dei dati**, essenziali per migliorare l'accuratezza del modello.
La fase di validazione infatti consente al modello di prendere decisioni

più accurate, grazie all'ottimizzazione delle scelte dovute ai dati di validazione.

Al contrario, il modello con le presentazioni peggiori è stato il *K Fold 3*, che ha ottenuto i seguenti risultati:

- **Accuracy**: 80.84467%
- **Precision**: 89.393%
- **Recall**: 80.845%
- **F1 Score**: 82.219%

6.6.3 K-Fold o Stratified K-Fold?

Avendo appreso la tecnica migliore da adottare per l'addestramento di un modello, ora analizziamo la media delle metriche tra **K Fold e Stratified K Fold.**

Come si puó notare dalle figure [27][28][29][30] a fine paragrafo, osserviamo che i modelli addestrati tramite la tecnica dello Stratified K-Fold presentano metriche di prestazione superiori rispetto ai modelli addestrati con il metodo K-Fold.
Questa differenza suggerisce che l'uso di tale tecnica, che assicura una suddivisione bilanciata delle classi nei vari fold, **contribuisce a migliorare l'efficacia di addestramento.**

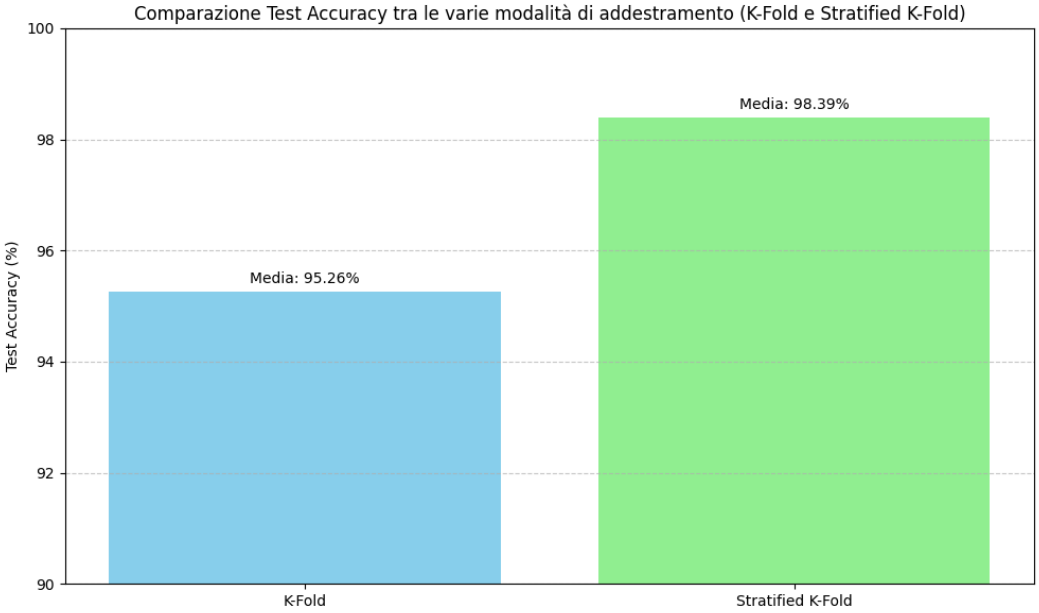

Figura 6.19: Comparazione media Test Accuracy

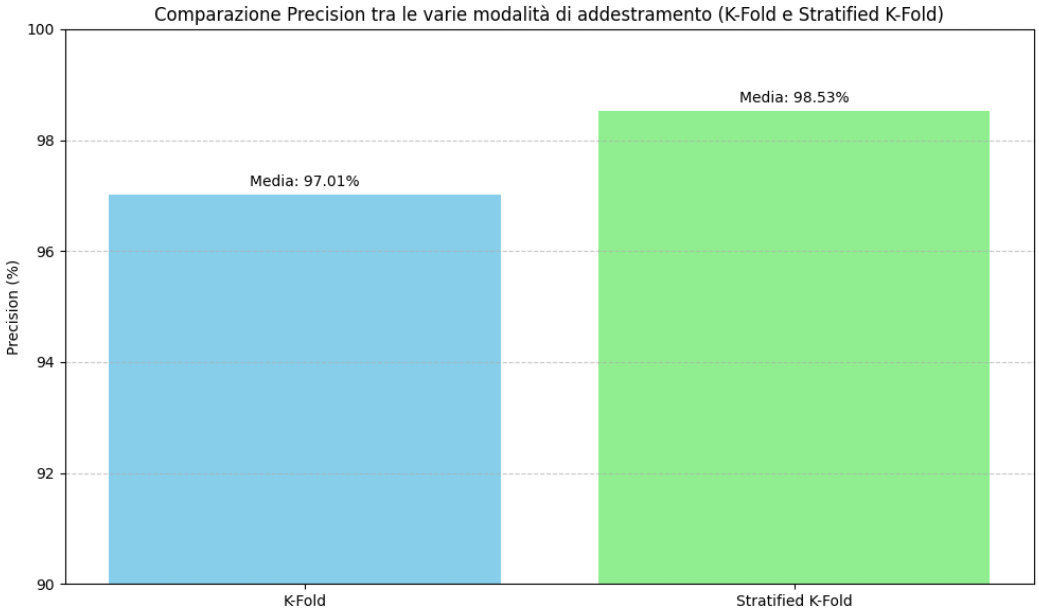

Figura 6.20: Comparazione media Precision

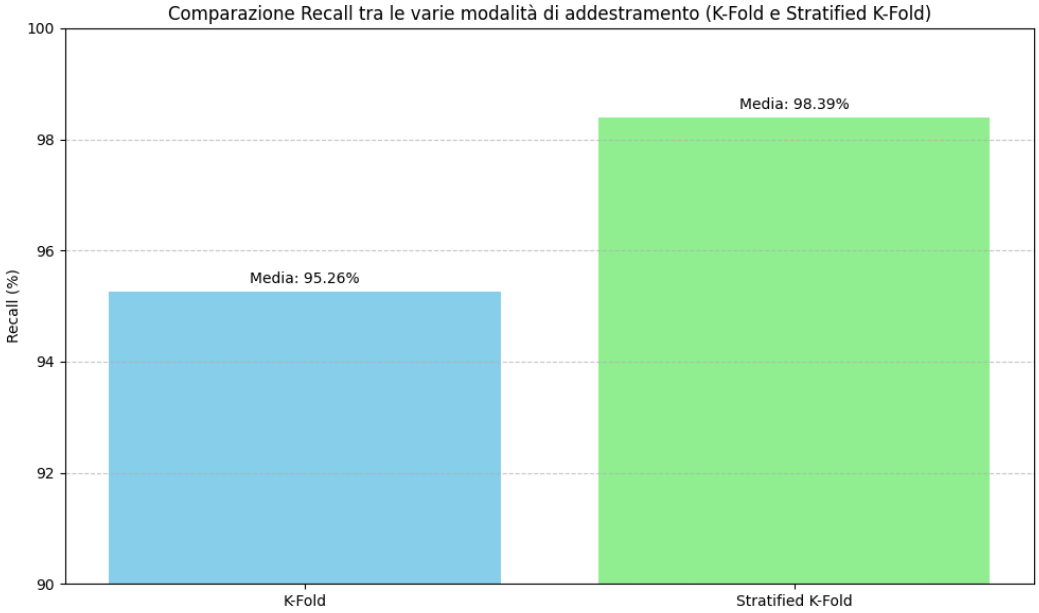

Figura 6.21: Comparazione media Recall

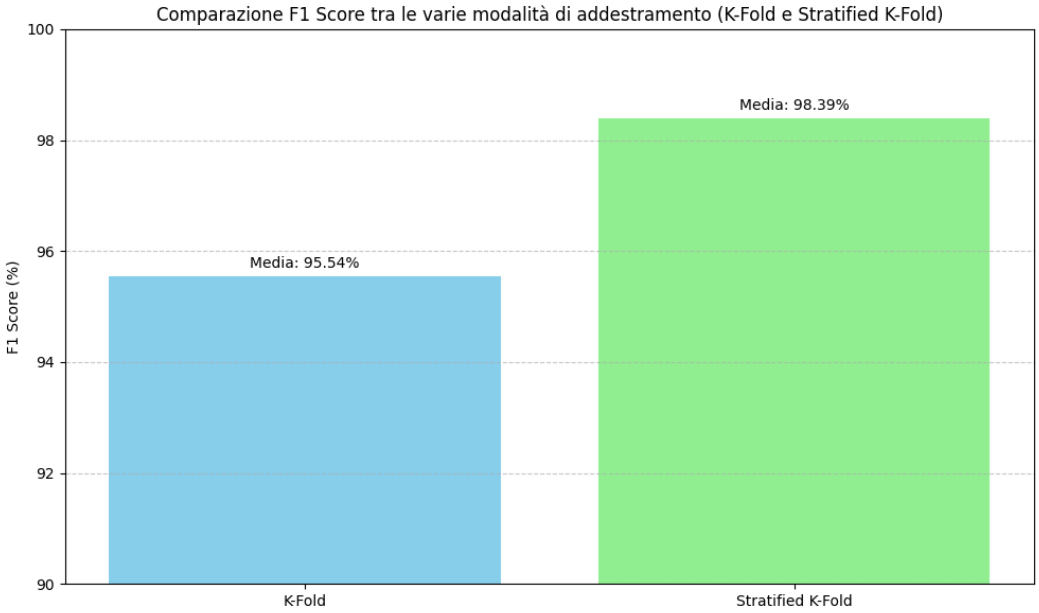

Figura 6.22: Comparazione media F1 Score

6.7 Confronto con altri modelli

In questa sezione confronteremo le prestazioni dei nostri tre migliori modelli per la rivelazione delle PII con quelle degli altri modelli pubblicati in letteratura scientifica.

I modelli selezionati sono:

- **Training-Validation-Test**
- **K-Fold 5**
- **Stratified K-fold 4**

La prima comparazione viene effettuata con il paper ***"Can large language models detect PII in code?"***[17]

In questo articolo vengono valutate le prestazioni di vari modelli NLP per il rivamentro delle PII, tramite classificazione binaria, stesso oggetto della seguente tesi.
Sono oggetto di analisi i seguenti modelli: **Mistral7B, CodeLlama 7B, Long Coderbase, GraphCodeBERT, CodeT5-220m, CodeT5Plus-770m, CodeBERT**.

Il dataset utilizzato per l'addestramento fa riferimento ai criteri di privacy definiti dal *California Consumer Protection Act (CCPA)* e del *General Data Protection Regulation(GDPR)* dell'Unione Europea.

Sono incluse le seguenti categorie di dati:

- ID univoco del dispositivo
- Identificatore dell'account o dell'individuo
- Dati demografici Internet
- Informazioni commerciali o finanziarie
- Dati biometrici multimediali
- Informazioni sull'impiego

– Informazioni educative

– Informazioni sulla posizione

Il dataset comprende dati provenienti da repository pubblici e da progetti open-source.
La selezione del dataset inoltre é stata ulteriormente verificata tramite un controllo manuale dal team di ricerca.
Inoltre, grazie all'ausilio di vari modelli NER, sono state identificate le frasi contenenti PII e le loro categorie.

Le comparazioni sono le seguenti.

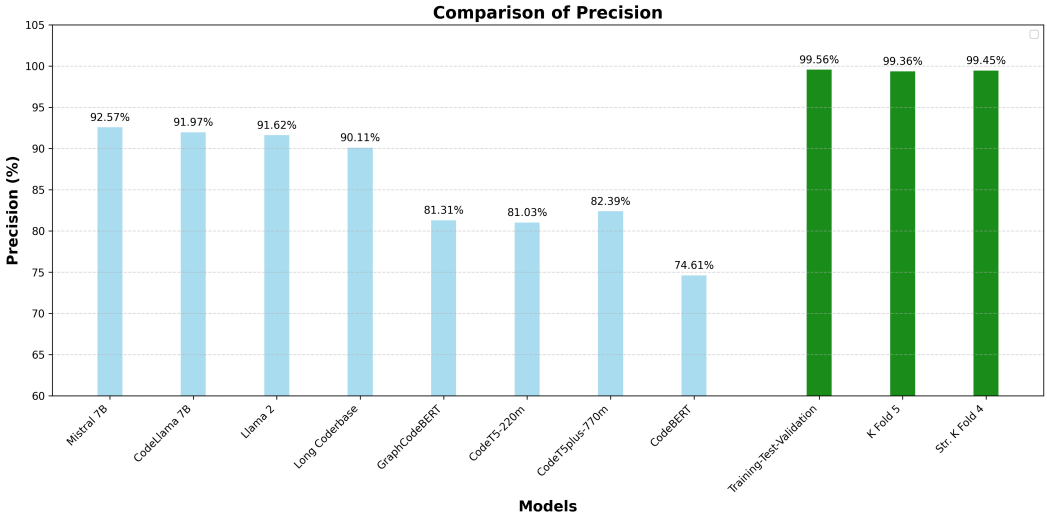

Figura 6.23: Comparazione delle precisione

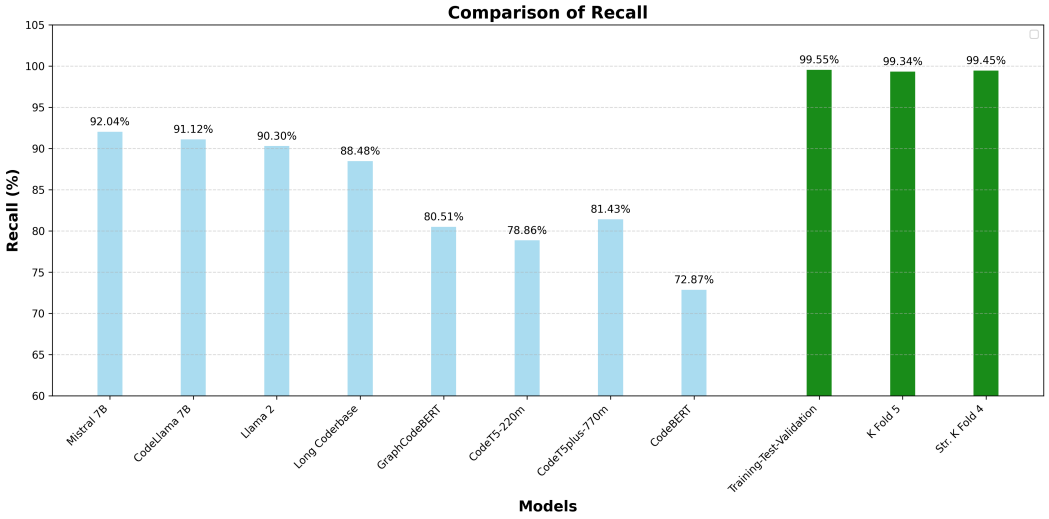

Figura 6.24: Comparazione del recall

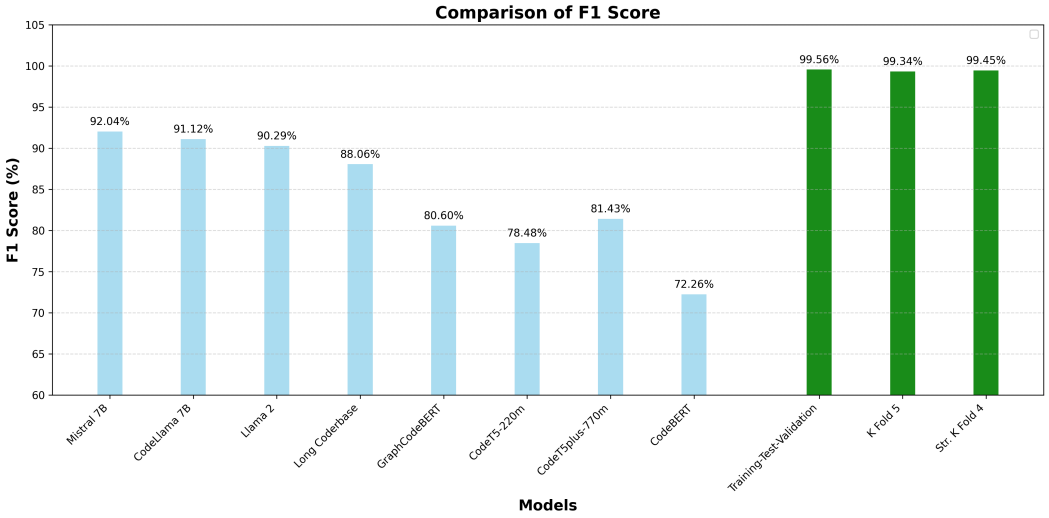

Figura 6.25: Comparazione dell'F1-Score

Nonostante le comparazioni sono state effettuate tra modelli addestrati utilizzando dataset differenti, i risultati evidenziano un miglioramento significativo delle prestazioni rispetto agli altri presi in considerazione.

La seconda comparazione viene effettuata con il modello presente nel paper ***"De-Identifying Student Personally Identifying Information with GPT-4"***[18]

In questo paper viene analizzato come **GPT-4** riesca ad identificare le informazioni personali degli studenti in un contesto scolastico.
Il dataset di testing utilizzato consiste in un raccolta di post su forum di studenti iscritti in nove diversi corsi online tra il 2012 e 2015.
Inizialmente i dati presi sono stati 500 post per ciascuno dei 9 corsi.
Successivamente sono stati esclusi i post non scritti in inglese o comprendenti solo link o caratteri speciali, ottenendo un dataset finale composto da 3.505 post da 2.882 studenti diversi.
I testi originali dei post sono stati mantenuti, inclusi i post contenenti errori ortografici e grammaticali, per valutare le performance del modello anche in condizioni linguistiche non ottimali.
I campi presi in considerazione sono:

- Economia
- Matematica
- Design
- Gamification
- Business trends
- Poesia
- Mitologia
- Probabilità
- Vaccini

Le metriche utilizzate nel paper per valutare le prestazioni del modello sono **Precision e Recall** e le nostre comparazioni si baseranno su esse. Nell'articolo, le valutazioni sono state eseguite per ogni corso, pertanto abbiamo considerato la media delle metriche (presente nel paper) per poter effettuare una comparazione significativa.

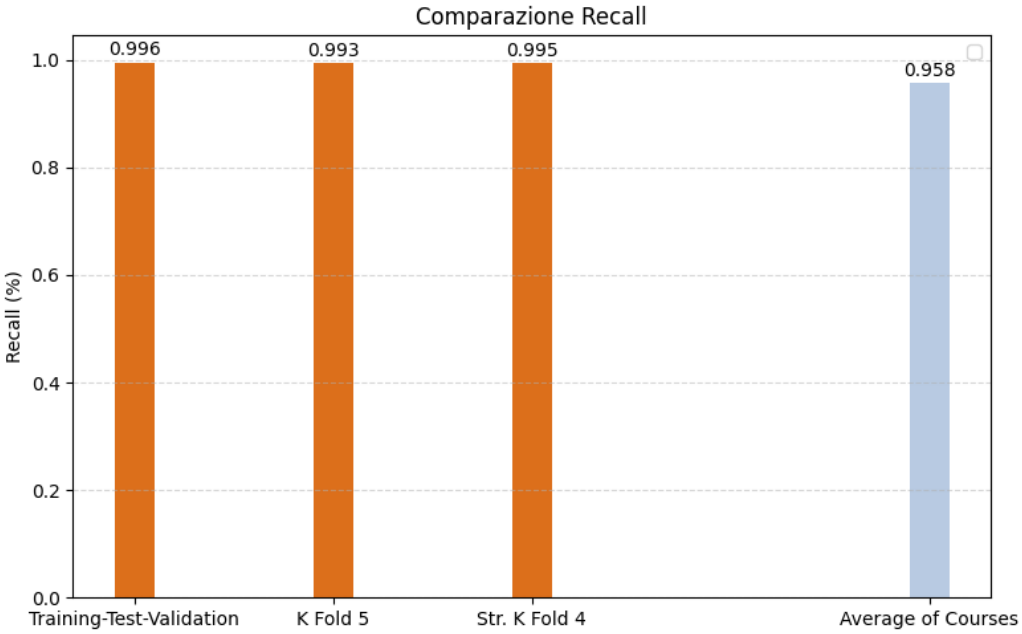

Figura 6.27: Comparazione Recall

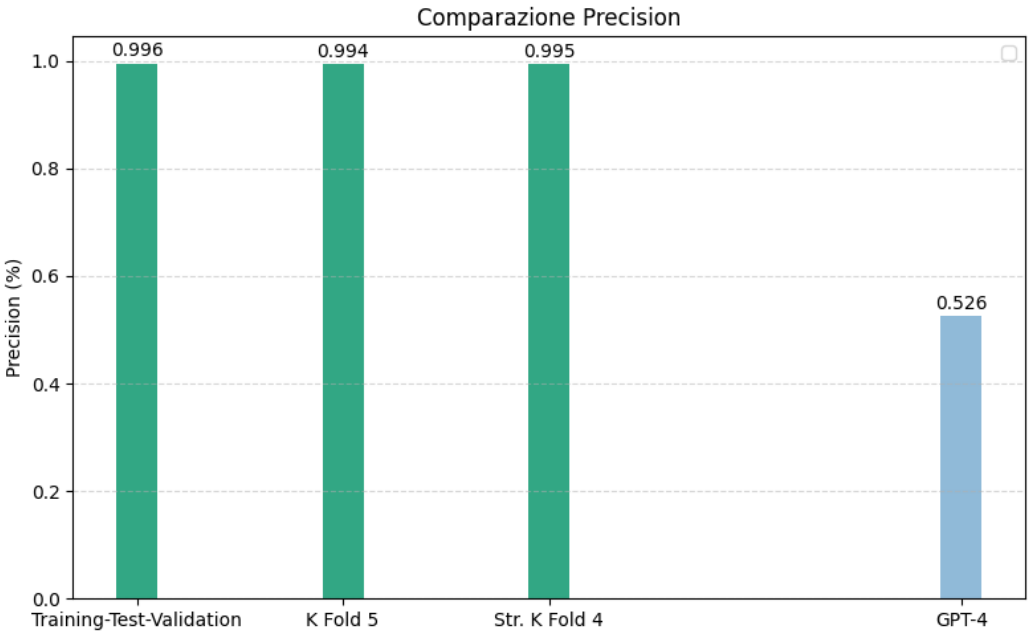

Figura 6.26: Comparazione Precisione

Dai risultati delle comparazioni, si evidenzia che, per quanto riguarda la **Precision**, i nostri modelli hanno ottenuto delle performance nettamente superiori rispetto a GPT-4.
È importante ricordare che la precisione rappresenta la capacità del modello di identificare le previsioni corrette effettuate.

Invece, esaminando il **Recall,** si osserva che i risultati di GPT-4 non si distaccano significativamente rispetto ai nostri modelli, dimostrando che tale modello é valido per la rivelazione delle PII, in quanto capace di riconoscere i veri positivi(TP) con un tasso particolarmente elevato.

Capitolo 7

Conclusioni

In conclusione, questo lavoro di tesi ha mostrato come sia possibile l'uso di modelli pre-addestrati BERT per il rilevamento delle informazioni personali nel contesto della protezione dei dati.

I risultati ottenuti dimostrano che il modello sviluppato ha raggiunto prestazioni elevate, superando gli altri presenti in letteratura.
Questo è stato raggiunto grazie ad un'analisi dettagliata del dataset di addestramento, ottenuto tramite l'ausilio di dataset esterni di alta qualità.

Un aspetto chiave di questo lavoro è stata la comparazione con le metriche di altri modelli NLP, come GPT-4, confermando le nostre ipotesi di lavoro, cioè che BERT , anche se è stato il pioniere del machine learning, è ancora ottimo per il rilevamento delle PII.

7.1 Limitazioni

Le principali limitazioni si concentrano su 3 aspetti.

- **Scelta del dataset**
- **Scelta del modello**
- **L'utilizzo di un singolo modelllo**

Per quanto riguarda il **dataset**, anche se appurata la sua ottima qualità, può essere migliorato ulteriormente.

Consultando la letteratura scientifica , notiamo che i dataset sono stati controllati da interi team di ricerca multidisciplinari, dopo lunghi periodi di lavoro e revisione.
Questo fa sì che l'accuratezza del dataset è un elemento fondamentale, se non il più importante, per l'addestramento di modelli. Un team dedicato avrebbe apportato degli ulteriori miglioramenti in aggiunta agli ottimi risultati ottenuti.

La **scelta del modello** rappresenta un'altra possibile area di miglioramento.
Sebbene BERT si sia dimostrato efficace, esistono modelli più recenti come RoBERTa, DeBERTa e varianti di BERT come CodeBERT.
Anche se alcuni modelli sono stati ampiamente analizzati nella tesi, potrebbero rappresentare alternative valide per il rilevamento delle PII.
Inoltre l'architettura stessa del modello potrebbe essere migliorata integrando tecniche di addestramento più avanzate come LoRa e QLora, che permettono una migliore gestione delle risorse mantenendo le alte prestazioni.

Un'altra limitazione è legata all'**utilizzo di un singolo modello** per il rilevamento delle PII. Per un approccio più robusto, è necessario un modello aggiuntivo di tipo NER.
Questo modello, posizionandolo in successione al nostro, potrebbe migliorare il rilevamento di informazioni personali, andando a identificarle e a classificarle.

7.2 Sviluppi Futuri

Gli sviluppi futuri si concentrano sulle limitazioni elencate nel paragrafo precedente.

Un'area di miglioramento riguarda l'integrazione di **modelli più avanzati** e di nuova generazione.
Questi modelli potrebbero fornire prestazioni superiori rispetto a quelle ottenute nel nostro studio in quanto, grazie alle loro architetture ottimizzate ed una capacità di generalizzazione superiore, sarebbero adatti per il rilevamento delle PII.

Un'ulteriore sviluppo del lavoro potrebbe prevedere l'integrazione di **modelli NER**, come *Spacy*, in grado di operare in sincronia con il modello BERT.
Questo approccio ci permetterebbe un rilevamento delle PII più efficace e di trattare anche casi limite che potrebbero sfuggire utilizzando un solo modello.

Per quanto riguarda il **dataset di addestramento**, un potenziale miglioramento sarebbe di aggiornarlo in modo che ci siano più tipologie di conversazioni. Inoltre una più accurata analisi manuale dei dati potrebbe contribuire ad ottenere un modello con prestazioni ancora più elevate, riducendo errori dovuti ad incongruenze dei dati.

Infine un ultimo aspetto riguarda **la suddivisione dei dati**.
Come evidenziato nel capitolo 6, il Training-Validation-Test è stata la tecnica che ha prodotto il modello più efficace.
Implementando un ulteriore fase di validazione, dopo l'addestramento dei modelli ottenuti tramite lo Stratified K-Fold, ci permetterebbe di ottenere un modello con prestazioni nettamente elevate, garantendo nei dati di test un corretto bilanciamento delle classi.

Elenco delle figure

Bibliografia

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[2] Mehreen Saeed. A gentle introduction to positional encoding in transformer models, part 1, 2023. Accessed: 2023-08-03.

[3] Josep Ferrer. How transformers work, 2024. Accessed: 2024-08-02.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.

[5] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. 2020.

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. 2018.

[7] Saeed Damadi, Golnaz Moharrer, and Mostafa Cham. The backpropagation algorithm for a math student. 2023.

[8] AI4Privacy. Pii masking 200k, 2023. Accessed: 2023-09-20.

[9] Langdon Holmes, Jiahe Wang, Scott Crossley, and Weixuan Zhang. The cleaned repository of annotated personally identifiable information.

[10] Reinhard Grabler, Matthias Hirschmanner, Helena Anna Frijns, and Sabine Theresia Koeszegi. Privacy agents: Utilizing large language models to safeguard contextual integrity in elderly care.

[11] Yuxin Wen, Leo Marchyok, Sanghyun Hong, Jonas Geiping, Tom Goldstein, and Nicholas Carlini. Privacy backdoors: Enhancing membership inference through poisoning pre-trained models, 2024.

[12] Md Rafi Ur Rashid, Jing Liu, Toshiaki Koike-Akino, Shagufta Mehnaz, and Ye Wang. Forget to flourish: Leveraging machine-unlearning on pretrained language models for privacy leakage, 2024.

[13] Kaggle. Generic sentiment multi-domain sentiment dataset, 2023. Accessed: 2023-09-20.

[14] Hugging Face. pii-masking-43k, 2023. Accessed: 2023-09-25.

[15] Kaggle. Simple dialogs for chatbot, 2023.

[16] Kaggle. Imdb dataset of 50k movie reviews, 2023.

[17] Ambarish Aniruddha Gurjar, L Jean Camp, Xinyao Ma, Chaora Anesu, and Tatiana Ringenberg. Can large language models detect pii in code? *Preprint*, 2023.

[18] Shreya Singhal, Andres Felipe Zambrano, Maciej Pankiewicz, Xiner Liu, Chelsea Porter, and Ryan S. Baker. De-identifying student personally identifying information with gpt-4. *Preprint*, 2023.