UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Fondamenti di Intelligenza Artificiale

Professore: Fabio Palomba
Alunno: Simone Cava
(https://github.com/SimoneC24/text-spam-checker)

Anno Accademico 2022-2023

Indice

1 Introduzione

1.1 Cos'è Text-Spam-Checker

Text-Spam-Checker è un progetto di controllo anti spam di testo. Come capita, a quasi tutti noi, spesso ci troviamo a ricevere mail, chiamate o messaggi che consideriamo "spam".

In generale lo spam è l'invio di messaggi pubblicitari e/o truffaldini che l'utente non ha richiesto. E' un fenomeno mondiale, complesso, che colpisce tanti utenti. Purtroppo non è facile difendersi o bloccarlo. I provider lo combattono ogni giorno lavorando, in sintesi, su tre filoni che operano insieme: i servizi internazionali antispam, i propri sistemi antispam, le segnalazioni degli utenti, che possono accelerare il lavoro.

L'obiettivo degli spammer è la pubblicità: comuni offerte commerciali, proposte di vendita di materiale pornografico o illegale, farmaci senza prescrizione medica. Il loro scopo è carpire dati personali, indirizzi email e password di utenti, numeri di carte di credito e di conto corrente ecc.

Gli spammer sono, a tutti gli effetti, dei criminali. Lo spam è soprattutto strumento di truffa, che propone improbabili progetti finanziari, o vincite false fatte dall'utente in questione, chiede le credenziali di accesso al tuo conto corrente online. Altri messaggi truffa imitano quelli del tuo provider, di tuoi fornitori o della tua banca, avvisandoti della "scadenza" di un servizio o del "mancato pagamento" di una fattura e, con la scusa dell'urgenza, chiedono di fare versamenti o di inviare IBAN, carte di credito o dati personali.

Gli spammers inviano le mail pubblicitarie o truffaldine a migliaia di indirizzi, utilizzando server in località remote o caselle mittenti create appositamente di volta in volta. Oppure servendosi di caselle reali di ignari utenti, che sono riusciti a violare grazie al fatto che il titolare utilizzava una password non sicura o che le credenziali sono finite in rete a causa dell'hackeraggio di qualche piccolo sito cui l'utente si era iscritto.

1.2 Le motivazioni

Va detto che l'attività di contrasto allo spam operata dai provider avviene in tempo reale, ma purtroppo non è possibile "bloccare" a priori lo spam per una serie di motivi. Gli spammers utilizzano server e email che cambiano di continuo; anche il formato dei messaggi cambia: mittente, oggetto, contenuto, formattazione e allo stesso modo vengono cambiate le proprietà dei messaggi. Inoltre le campagne possono partire all'improvviso con un numero non elevato di messaggi, spediti però da una molteplicità di indirizzi diversi.

Il progetto Text-Spam-Checker nasce dall'idea di evitare queste truffe. Il concetto di spam è un qualcosa con cui tutti noi abbiamo a che fare ogni giorno anche in ambienti dove questo concetto non sembra esistere. Abbiamo pensato quindi di usare l'Intelligenza Artificiale per ammortizzare questo problema.

2 Specifiche P.E.A.S.

- **P** erformance: sono le misure di prestazione adottate per valutare l'operato di un agente, in questo caso valutiamo se l'agente restituisce il valore giusto tra spam e non-spam. Per valutare la bontà delle predizioni ho usato una matrice di confusione, anche detta tabella di errata classificazione, la quale restituisce una rappresentazione dell'accuratezza del classificatore.
- **E** nvironment: Descrizione degli elementi che formano l'ambiente. L'agente in questione opera nel campo dei messaggi spam. In seguito abbiamo descritto tutte le caratteristiche inerenti all'ambiente in questione.
- **A** ctuators: Gli attuatori servono all'agente per compiere le azioni. In questo caso sarà il modello di machine learning addestrato.
- **S** ensors: I sensori sono dispositivi attraverso i quali l'agente riceve gli input percettivi. Nel nostro caso sarebbe la tastiera attraverso la quale scrive il messaggio.

2.1 Caratteristiche dell'ambiente

- **Singolo agente**: nell'ambiente è possibile avere solo un agente.
- **Tipo di ambiente**: classificatore di messaggi spam o non-spam.
- **Completamente osservabile**: attraverso i sensori l'agente conosce sempre l'ambiente. Ha una visione completa dei messaggi all'interno del database.
- **Non deterministico**: è un ambiente che non cambia nel tempo. Lo stato successivo dell'ambiente non è determinato dallo stato corrente e dall'azione eseguita dall'agente.
- **Episodico**: L'esperienza dell'agente è divisa in "episodi" atomici, dove ciascun episodio consiste dell'eseguire una singola azione. La scelta dell'azione in ciascun episodio dipende dall'episodio stesso.
- **Statico**: l'ambiente resta invariato mentre l'agente esegue le azioni.
- **Discreto**: l'ambiente fornisce un numero limitato di percezioni e azioni distinte.

3 Machine Learning

Il Machine Learning è una branca dell'Intelligenza Artificiale, al contrario di quello che si pensa comunemente con il termine Machine Learning non intendiamo l'intero concetto di Intelligenza Artificiale ma appunto solo una parte.

Attualmente, il Machine Learning è utilizzato ovunque. Quando interagiamo con le banche, acquistiamo online o utilizziamo i social media, vengono utilizzati gli algoritmi di Machine Learning per rendere la nostra esperienza efficiente, facile e sicura. Il Machine Learning e la tecnologia associata si stanno sviluppando rapidamente e noi abbiamo appena iniziato a scoprire le loro funzionalità. Questo è uno dei tanti motivi per cui la scelta è ricaduta su un agente capace di apprendere e non su un agente basato su utilità.

Dunque il Machine Learning è sostanzialmente lo studio che esplora algoritmi e tecniche in grado di apprendere dai dati a loro disposizione e sulla base di questi fare predizioni.

Ci sono vari tipi di apprendimento quando parliamo di Machine Learning. Apprendimento supervisionato, semi-supervisionato, non supervisionato e per rinforzo.
Per il nostro progetto ci siamo concentrati sull'apprendimento supervisionato. In questo progetto, quindi, si userà un algoritmo che usa dati etichettati, ovvero dati di cui si conosce la variabile dipendente che sarebbe il valore che il modello dovrà predire una volta addestrato.

4 CRISP-DM

Sono due le cose principali a cui pensare quando si vuole progettare una soluzione basata su Machine Learning; data & software engineering. Il modello CRISP-DM rappresenta il ciclo di vita di progetti basati su Intelligenza Artificiale e Data Science.

CRISP-DM è l'acronimo di **CR**oss-**I**ndustry **S**andard **P**rocess for **D**ata **M**ining. Possiamo paragonare il modello CRISP-DM ad un modello a cascata con feedback utilizzato per lo sviluppo di sistemi software tradizionali. Il CRISP-DM è un modello non sequenziale in cui le diverse fasi possono essere eseguite un numero illimitato di volte. Di seguito eseguo e spiego tutte le diverse fasi.

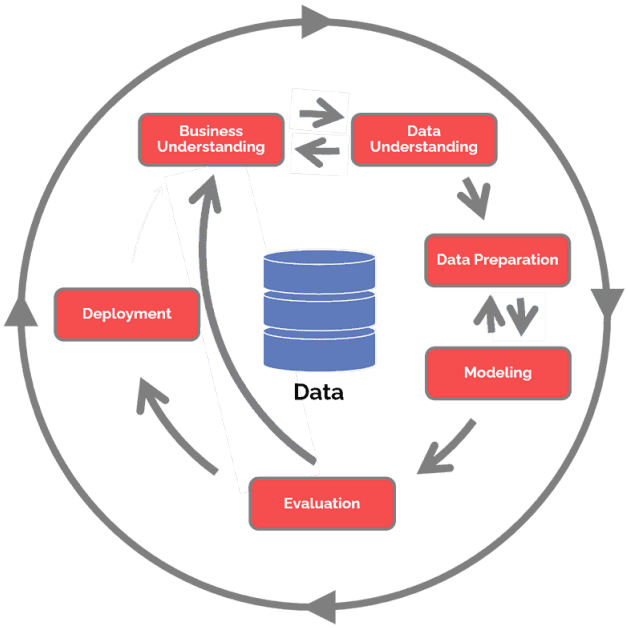

Figura 1: crisp-dm

4.1 Business Understanding

La prima fase è chiaramente quella di raccolta dei requisiti e di definizione degli obiettivi di business che si intende raggiungere (ovvero, che cosa deve fare il machine learner che stiamo progettando).

La fase di business understanding prevede la definizione dei cosiddetti business **success criteria**, ovvero i criteri secondo i quali potremo accertare che il sistema costruito è in linea con gli obiettivi di business.

In questa fase, bisogna inoltre determinare la disponibilità delle risorse, stimare i rischi, definire i relativi piani di contingenza e condurre una analisi costi-benefici. Oltre che definire i criteri di successo da un punto di vista di business, è inoltre necessario definire gli obiettivi tecnici che si intendono raggiungere. Infine, verranno selezionate le tecnologie ed i tool necessari agli obiettivi.

- **Obiettivi di business**: Il machine learner che intendiamo progettare avrà l'obiettivo di stimare o predire il valore della variabile dipendente di un messaggio che come abbiamo detto può essere spam o ham (non-spam) .
- **Risorse disponibili**: Per raggiungere l'obiettivo ci serviremo di un dataset di messaggi. Il dataset in questione consta di circa 5500 righe, in formato csv, e in ogni riga troviamo una stringa di testo, la quale rappresenta il messaggio da valutare, e un'etichetta in formato testuale autoesplicativa, con due valori ammessi (spam, ham). Facendo alcune ricerche, anche se questo dataset non ha tutte le occorrenze di messaggi che servirebbero, è sembrato il più adatto per lo sviluppo del machine learner. Abbiamo potuto scaricare il suddetto dataset da Kaggle.
- **Stima dei rischi**: Il dataset a nostra disposizione non ha abbastanza occorrenze per permettere al modello un apprendimento quasi perfetto. In compenso tutte le istanze del dataset hanno una qualità elevata in termini di lunghezza e studio del messaggio.
- **Tecnologie e strumenti**: Per acquisire, analizzare e modellare i dati abbiamo utilizzato il linguaggio Python poiché ho potuto fare uso di alcune librerie. Le librerie e gli aiuti di maggiore rilevanza sono:
  - Pandas che fornisce strutture e strumenti per l'analisi dei dati;
  - Seaborn per la creazione di grafici;
  - Basic Text Classification una guida di Tensorflow su come realizzare un modello di classificazione del testo.
  - Save and load model una guida di Tensorflow su come realizzare dei modelli persistenti.
  - Matrice di confusione una guida su come mostrare una matrice di confusione in un ambiente Python.

4.2 Data Understanding

Sulla base degli obiettivi definiti nella fase precedente, il secondo passo consiste nell'identificazione, collezione e analisi dei dataset che possono portare al raggiungimento degli obiettivi.

Innanzitutto, quindi, vengono acquisiti i dati necessari al raggiungimento degli obiettivi di business e i dati verranno poi caricati in un tool di analisi dei dati. I dati vengono quindi esaminati e documentati rispetto al loro formato, il numero di record e il significato di ciascun campo.

Il terzo task consiste nell'esplorazione dei dati: questi vengono visualizzati e, cosa più importante, eventuali relazioni vengono identificate. Infine, il processo di qualità dei dati, vengono identificati e documentati possibili problemi di qualità dei dati (ad esempio, dati mancanti).

Nel nostro caso il dataset in questione ha due colonne e circa cinquemilacinquecento righe. La prima colonna riguarda le etichette che come abbiamo già detto possono assumere due valori ovvero ham o spam. La seconda colonna contiene i messaggi che sono scritti in lingua inglese e contengono un massimo di novecentodieci caratteri.

Per leggere il dataset abbiamo usato una funsionalità della libreria *Pandas*, il dataset si presenta nella seguente forma:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

Figura 2: dataset

Dalla tabella delle frequenze possiamo individuare in modo chiaro quali sono le etichette e i messaggi più frequenti:

	count	unique	top	freq
Category	5572	2	ham	4825
Message	5572	5157	Sorry, I'll call later	30

Figura 3: tabella frequenze

Infine abbiamo potuto anche visualizzare la distribuzione delle etichette:

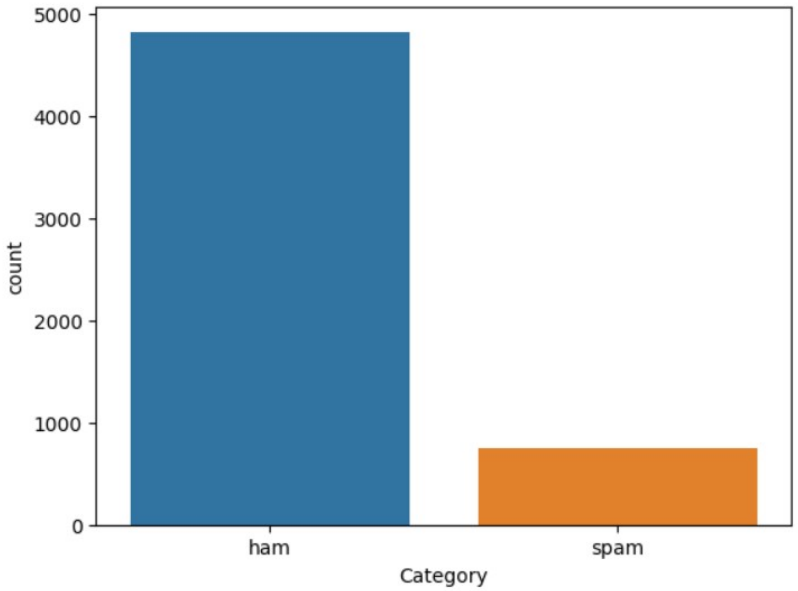

Figura 4: distribuzione etichette

Come si può osservare nel grafico a barre riportato di sopra, la distribuzione delle etichette è particolarmente sbilanciata, questo potrebbe portare a problematiche nelle fasi successive all'apprendimento.

4.3 Data Preparation

L'obiettivo di questa fase è quello di preparare i dati in maniera tale che possano essere utilizzati nei successivi passi del processo.

Questo include un processo fondamentale che è noto come feature engineering, ovvero la selezione delle caratteristiche del problema che hanno maggiore potenza predittiva. Inoltre, questa fase include l'implementazione dei processi di pulizia dei dati sulla base dei problemi di qualità riscontrati nella fase precedente.

Sulla base della pulizia fatta così come dell'analisi della potenza predittiva delle caratteristiche considerate, il progettista può considerare di estendere le caratteristiche da considerare.

Infine, i dati vengono formattati in una maniera tale che possano essere presi in input da un modello di machine learning, questo potrebbe dipendere dai tool selezionati in fase di business understanding.

La data preparation si divide in 4 fasi fondamentali che sono quelle descritte di seguito.

4.3.1 Data cleaning

La prima fase è quella di pulizia dei dati. Vista la distribuzione sbilanciata delle etichette del dataset, si potrebbe pensare di voler svolgere una fase di data imputation, ma come vedremo successivamente, questa distribuzione NON sarà un problema per il nostro modello.
Inoltre, potrebbe essere necessario procedere con una fase di pulizia che riguarderebbe gli outlier rispetto alla distribuzione per conteggio di parole. Ovvero, ci siamo chiesti se fosse necessario eliminare quei pochi messaggi che contengono un alto numero di parole e che quindi abbiamo considerato outlier.

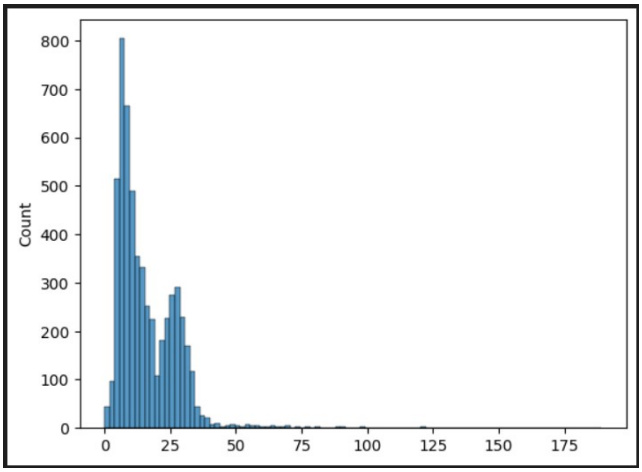

Figura 5: distribuzione per conteggio di parole

Lo svolgimento di questa fase si renderebbe necessaria solo nel momento in cui, nelle fasi successive all'addestramento, trovassimo una correlazione tra errori commessi dal modello e la lunghezza dei messaggi.
Per controllare se ci fosse questa correlazione abbiamo creato un grafico degli errori che ci permette di capire quando e come il nostro modello sbaglia. Il suddetto grafico è riportato di seguito.

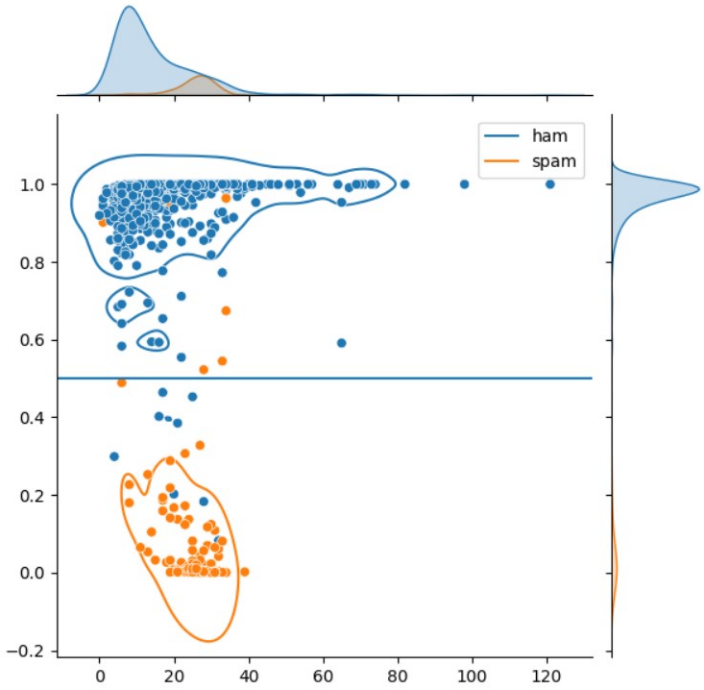

Figura 6: grafico errori rete neurale

Guardando il grafico ci accorgiamo che gli sbagli del modello non sono concentrati dove i messaggi sono più lunghi. Quindi, considerando che non abbiamo trovato nessuna correlazione tra la distribuzione per conteggio di parole e gli errori fatti dal nostro modello, abbiamo scelto di NON eseguire la pulizia per quanto riguarda gli outlier. Si precisa che potrebbe comunque esistere una caratteristica del dataset, non ancora individuata, associabile alla presenza di errori.
Un'attività che, invece, abbiamo eseguito per la fase di pulizia dei dati è quella di sostituire ogni cifra presente nei messaggi con il simbolo "#", in modo tale che possano essere ignorate in fase di

tokenizzazione. Questa attività è stata necessaria per eliminare dati rumorosi dal nostro dataset, i numeri non sono fondamentali per il nostro scopo perché, anche se in un messaggio dato in input al modello ci sarà un numero, è molto difficile che questo numero sia riconosciuto dal modello e quindi già esistente nel dataset.

4.3.2 Feature scaling

La seconda fase è il feature scaling che sono un insieme di tecniche che consentono di normalizzare o scalare l'insieme di valori di una caratteristica.

Nel nostro progetto è presente testo scritto in linguaggio naturale. Abbiamo una tecnica che ci permette di normalizzare il testo. L'analisi del linguaggio naturale è utilizzata in molti contesti e, non a caso, un intero campo dell'Intelligenza Artificiale è dedicato al **N**atural **L**anguage **P**rocessing. Indipendentemente dal contesto, un comune task di NLP segue alcuni step.

Nel nostro progetto avevamo il problema di trasformare testo, espresso in linguaggio naturale, in token numerici comprensibili e interpretabili dal modello.

La tokenizzazione è il processo in cui la frase viene divisa in token, ossia unità del periodo più piccole sulle quali si andrà a lavorare singolarmente. Un token è, generalmente una sotto stringa della frase o una piccola parola. Tale processo può essere effettuato seguendo diverse strategie, come il riconoscimento dei *white space* o particolari segni di punteggiatura. Nel nostro caso il modello ha memorizzato ogni singolo token come numero, inoltre, dobbiamo aggiungere che, la tokenizzazione è stata effettuata solo sui dati di training. In questo modo non avendo memorizzato i token presenti nei dati di test potremo avere un test veritiero.

In supporto alla Tokenizzazione, giunge in aiuto l'API di tokenizzazione **Tokenizer**, fornita da **TensorFlow**. Questa consente di vettorializzare un testo, trasformando ogni testo in una sequenza di interi, nel nostro caso ogni intero è l'indice di un token in un vocabolario.

Nella prima fase di utilizzo del nostro "tokenizzatore", andiamo a costruire il nostro vocabolario a partire dal dataset di apprendimento. Abbiamo potuto notare che il vocabolario a nostra disposizione consta di 7102 token. La taglia del vocabolario è, quindi, nulla rispetto alla gamma di terminologie usate realmente nell'ambiente della messaggistica. Per avere un vocabolario più ampio dovremmo aggiungere istanze di messaggi nel nostro dataset.

4.3.3 Feature selection

Per quanto riguarda il feature selection il progettista estrae le **caratteristiche principali** del problema. Identificare buone feature porta molti vantaggi pratici. Il feature selection è quindi il processo tramite il quale vengono selezionate le caratteristiche correlate al problema in esame, lo possiamo fare con l'**eliminazione di feature con bassa varianza** o **con l'eliminizaione univariata di feature**.

In questo progetto NON c'è stato bisogno di eseguire nessuna azione di questa fase poiché tutte le feature sono di tipo **categorico** e non numerico.

4.3.4 Data balancing

Molti problemi potrebbero essere sbilanciati. Con il data balancing andiamo a bilanciare il dataset con la tecnica dell'**undersampling**, che sarebbe l'eliminazione di alcune righe appartenti alla classe con un numero di elementi maggiore, oppure con la tecnica dell'**oversampling** possiamo aggiungere nuove righe per la classe che ha meno istanze.

Nel nostro progetto c'è un problema abbastanza importante di data balancing. Come abbiamo potuto vedere, nel grafico inerente alla distribuzione delle etichette, il nostro dataset è sbilanciato. Nonostante questo, non abbiamo eseguito undersampling, perché rischiavamo di avere troppe poche istanze nella totalità del dataset e questo ci portava a delle predizioni meno esatte. Non abbiamo nemmeno aggiunto righe per quanto riguarda i messaggi considerati spam perché non è stato possibile trovare messaggi adatti al nostro set di dati ed era troppo oneroso crearne di nuovi. Non potendo aggiungere e/o togliere righe dal nostro dataset abbiamo deciso di seguire il suggerimento dato durante la presentazione del progetto. Seguendo il suggerimento, andiamo a porre

un maggiore peso alla classe che risulta essere in minoranza, che nel nostro caso sarebbero i messaggi spam. La modalità di impostazione di questi pesi è stata derivata dalla documentazione di TensorFlow.

La scelta dei pesi è pienamente imputabile a valutazioni empiriche effettuate reiterando l'attività di addestramento, apportando di volta in volta modifiche ai diversi pesi, fino al raggiungimento di un risultato soddisfacente. In particolare abbiamo dato peso 1 ai messaggi considerati "spam" e 0.2 ai messaggi "ham".

4.4 Data Modeling

Una volta sistemati i dati, è ora di iniziare la fase di modellazione, che è sempre particolarmente complicata e "unica", nel senso che la definizione di un algoritmo di machine learning dipende strettamente dal problema in esame e dai dati a disposizione.

In primo luogo, va selezionata la tecnica o l'algoritmo da utilizzare: ad esempio, conviene modellare il problema come un problema di classificazione o regressione? Quale soluzione sarà più adatta e utile al raggiungimento degli obiettivi? Dopodiché, si passa alla fase di addestramento. In questo caso, si configurano i parametri del modello selezionato, si addestra il modello e si descrivono i risultati ottenuti in fase di addestramento.

Molto spesso, il progettista sarà costretto a tornare nella fase di data preparation per effettuare ulteriori operazioni sui dati.

Tra le architetture di apprendimento profondo si annoverano le **deep neural network**, che sono state applicate in molti campi come la visione artificiale, nel riconoscimento automatico del discorso, nell'elaborazione del linguaggio naturale, nel riconoscimento audio e nella bioinformatica. In termini pratici le reti neurali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare, è questo il motivo principale che ci ha portati a scegliere una rete neurale piuttosto che un altro classificatore. Una rete neurale artificiale riceve segnali esterni su uno strato di nodi di ingresso (unità di elaborazione), ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.Quindi possiamo dire che i livelli sono impilati in sequenza per costruire il classificatore.

4.4.1 Ma come funziona la nostra rete neurale?

Chiariamo prima di tutto cosa diamo in input alla rete neurale. Il tokenizzatore da noi addestrato nel momento in cui gli viene chiesto di tokenizzare un messaggio applica la tokenizzazione. In pratica, sostituisce a ogni token a lui noto individuato nel messaggio il numero corrispondente all'indice del vocabolario. Alla fine della fase di tokenizzazione avremo un array per ogni messaggio, nella quale, al posto delle parole abbiamo i token del nostro vocabolario, questi token rappresentano gli input che entreranno nella rete.
In particolare, nella rete neurale che abbiamo utilizzato per questo progetto il primo livello è un livello di **Embedding**. Questo livello prende i messaggi tokenizzati e cerca un vettore di incorporamento per ogni indice di parola. I vettori aggiungono una dimensione all'array di output. Dopo il primo livello sono presenti due livelli intermedi che sono chiamati livelli di Dropout. I livelli Dropout impostano in modo casuale le unità di input su 0 con una frequenza di **rate** (passata come unico argomento) a ogni passaggio durante la fase di addestramento, che aiuta a prevenire l'overfitting. Gli input non impostati su 0 vengono aumentati di 1/(1 - rate) in modo tale che la somma di tutti gli input rimanga invariata. Possiamo dire che il secondo livello della nostra rete ha un rate pari a 0.2 mentre il terzo possiede un rate pari a 0.4. Successivamente, un livello **GlobalAveragePooling1D** restituisce un vettore di output a lunghezza fissa per ogni esempio calcolando la media sulla dimensione della sequenza. Ciò consente al modello di gestire input di lunghezza variabile, nel modo più semplice possibile. Infine l'ultimo strato è densamente connesso con un singolo nodo di output.

Come nel nostro caso una rete neurale utilizza un algoritmo di **retropropagazione** dell'errore (backpropagation). Esso permette di modificare i pesi delle connessioni in modo tale che si mini-

mizzi una certa funzione errore **E**. Tale funzione dipende dal vettore h-esimo di output restituito dalla rete, dato il vettore h-esimo di ingresso e dal vettore h-esimo di output che noi desideriamo (che fa parte del training set). L'algoritmo di backpropagation può essere diviso in due passi:

- **Forward pass**: l'input dato alla rete è propagato al livello successivo e così via ai livelli successivi (il flusso di informazioni si sposta in avanti, cioè forward). Si calcola dunque E(w), l'errore commesso.
- **Backward pass**: l'errore fatto dalla rete è propagato all'indietro (backward) e i pesi sono aggiornati in maniera appropriata.

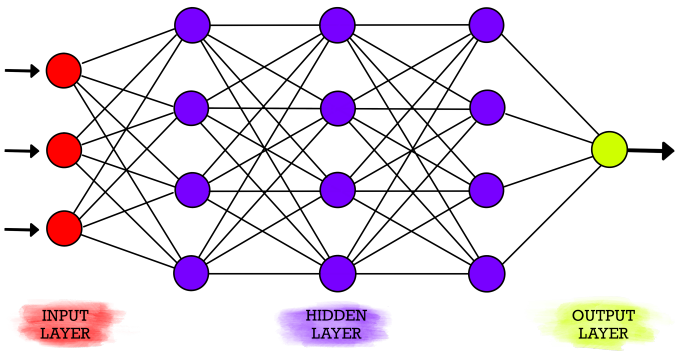

Figura 7: rete neurale

Inoltre, il modello che abbiamo addestrato per questo progetto, può essere definito come "binario" o "a due classi", ciò indica che potrà fornirci risposte di tipo 0 e 1 o nello specifico per il nostro caso, "HAM" e "SPAM".

Poiché si tratta di un problema di classificazione binaria e il modello emette una probabilità (uno strato di unità singola con un'attivazione sigmoidea).Infatti una delle scelte effettuate durante la fase di modeling, a grande impatto per le fasi successive, è quella di non arrotondare i valori di output della rete neurale al più vicino intero, dunque, sostituendo una più ovvia funzione di attivazione a passo unitario con una **funzione sigmoidea**, andando successivamente ad effettuare una arrotondamento con una funzione esterna alla rete.

Questa scelta è stata particolarmente influenzata dal vantaggio di poter, in un ambiente in cui il modello non dovrà essere distribuito ma valutato, utilizzare le classificazioni per effettuare valutazioni, non solo binarie, le quali ci fornirebbero solo informazioni del tipo "Classificazione Sbagliata" o "Classificazione Corretta", piuttosto avere anche una misura di quanto il modello sia sicuro della sua scelta. Inoltre la funzione sigmoidea ci consente di evitare che le valutazioni del modello si vadano a concentrare in valori vicini a 0.5, relativi a casi di "estrema indecisione".

Un grande aiuto per la costruzione del modello in questione è stato fornito dalle innumerevoli guide sull'utilizzo di TensorFlow, andando ad apportare piccole modifiche al codice fornito, basandosi su osservazioni fatte durante lo svolgimento delle diverse attività:
Basic text classification - TensorFlow Core.

4.4.2 C'è davvero bisogno di usare una rete neurale?

Alcuni degli algoritmi di classificazione del testo più popolari includono anche la famiglia di algoritmi Naive Bayes, che potrebbero essere una valida alternativa alla rete neurale da noi usata. La

famiglia di algoritmi statistici Naive Bayes sono alcuni degli algoritmi più utilizzati nella classificazione del testo e nell'analisi del testo, in generale. Uno dei membri di quella famiglia è Multinomial Naive Bayes (MNB) con un enorme vantaggio, ovvero che si possono ottenere risultati davvero buoni anche quando il set di dati non è molto grande e le risorse computazionali sono scarse. Naive Bayes si basa sul teorema di Bayes, che ci aiuta a calcolare le probabilità condizionali del verificarsi di due eventi, in base alle probabilità del verificarsi di ogni singolo evento. Quindi stiamo calcolando la probabilità di ogni tag per un dato testo, e poi emettiamo il tag con la probabilità più alta. La probabilità che A, se B è vero, è uguale alla probabilità che B, se A è vero, moltiplicata per la probabilità che A sia vero, divisa per la probabilità che B sia vero. Ciò significa che qualsiasi vettore che rappresenta un testo dovrà contenere informazioni sulle probabilità di apparizione di determinate parole all'interno dei testi di una data categoria, in modo che l'algoritmo possa calcolare la probabilità che quel testo appartenga alla categoria.

Il deep learning invece è un insieme di algoritmi e tecniche ispirati al funzionamento del cervello umano, chiamati reti neurali. Le architetture di deep learning offrono enormi vantaggi per la classificazione del testo perché si comportano con altissima precisione con ingegneria e calcolo di livello inferiore. Il deep learning è l'apprendimento automatico gerarchico, che utilizza più algoritmi in una catena progressiva di eventi. È simile a come funziona il cervello umano quando prende decisioni, utilizzando diverse tecniche contemporaneamente per elaborare enormi quantità di dati. Gli algoritmi di deep learning richiedono molti più dati di addestramento rispetto ai tradizionali algoritmi di machine learning (almeno milioni di esempi con tag). Tuttavia, non hanno una soglia per l'apprendimento dai dati di addestramento, come i tradizionali algoritmi di apprendimento automatico continuano a migliorare man mano che vengono alimentati con più dati.

La risposta al nostro quesito, quindi, è sicuramente si, potevamo usare ad esempio l'algoritmo della famiglia Naive Bayes, **Multinomial Naive Bayes(MNB)**, citato in precedenza. Nonostante ciò abbiamo pensato che per usi futuri e con l'aggiunta di dati al nostro dataset il comportamento dato dal modello che fa uso della rete neurale era più adatto. La scelta dei modelli relativi a questo progetto è stata particolarmente influenzata da letture di diversi articoli riguardo l'argomento, nello specifico: Text Classification: What it is And Why it Matters.

Di seguito valuteremo i risultati ottenuti usando il nostro modello e un modello che fa uso dell'algoritmo **MNB**.

4.5 Evaluation

La fase di validazione ha l'obiettivo di valutare se i risultati sono chiari, se sono in linea con gli obiettivi di business e se rivelano delle prospettive aggiuntive alle quali il progettista non aveva pensato.

In questa fase, è inoltre importante verificare la consistenza e la solidità dell'intero processo. Ad esempio, ci sono degli aspetti che non sono convincenti? Ci sono delle alternative metodologiche che potrebbero portare a risultati diversi? E come queste alternative impattano i risultati ottenuti?

Una volta avute le risposte, si può procedere alla definizione dei prossimi passi da effettuare. Possiamo considerare la definizione dell'approccio completa? Oppure è necessario fare un passo indietro e valutare opzioni diverse?

4.5.1 Andamento dei miglioramenti

Il seguente grafico mostra l'andamento dei miglioramenti ad ogni iterazione dell'addestramento che abbiamo eseguito. Per dei risultati soddisfacenti abbiamo deciso che trenta iterazioni sarebbero state abbastanza per l'addestramento dei nostri dati. Infatti come possiamo vedere dal grafico, che mostra sull'asse delle x il numero di iterazioni fatte dal modello e sull'asse delle y il valore che indica una penalità legata a una predizione sbagliata, c'è un netto miglioramento, anche se ci sono tante cose da precisare. La linea di colore arancione indica la funzione di perdita relativa ai dati di validazione che sarebbero una porzione dei dati di training. La linea blu, invece, indica la funzione di perdita per i dati di training.

Il nostro modello è stato addestrato, con dei risultati apparentemente molto soddisfacenti. Ciò però NON significa necessariamente che il suo comportamento sia quello da noi desiderato.

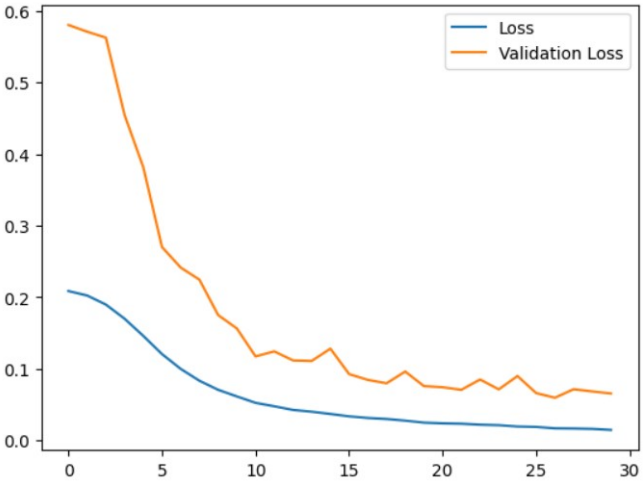

Figura 8: andamento miglioramenti rete neurale

4.5.2 Matrice di confusione

Si rende molto utile in questa fase di comprensione dei dati, l'utilizzo di una **matrice di confusione** per una rappresentazione dei *risultati ottenuti dal modello*. Calcoliamo quindi per ogni etichetta quante sono le entry per le quali le etichette sono state correttamente predette dal modello nel **dataset di test** e quali invece sono state sbagliate.

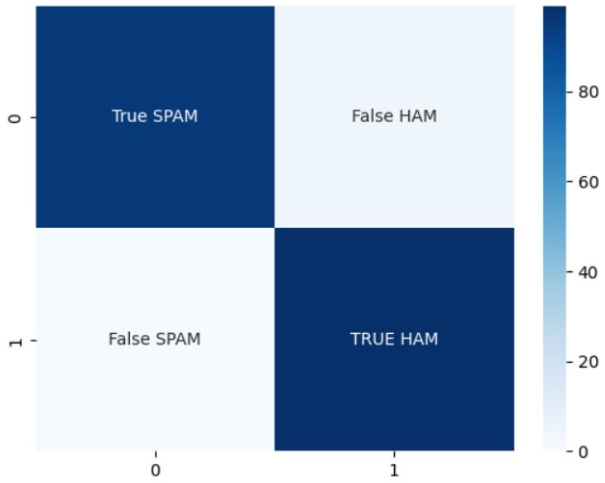

Figura 9: matrice di confusione rete neurale

Osservando la matrice di confusione possiamo definirci soddisfatti dei risultati ottenuti. Il nostro modello predice bene sia i messaggi spam che quelli non-spam che è proprio ciò che vogliamo.

4.5.3 Valutazione degli errori

Come vedremo dal grafico riportato di seguito il modello sbaglia poche volte e a differenza di quel che si poteva immaginare, la maggior frequenza degli errori è in corrispondenza delle frasi con un più basso conteggio di parole, fattore incentivato, probabilmente dalla loro elevata frequenza.

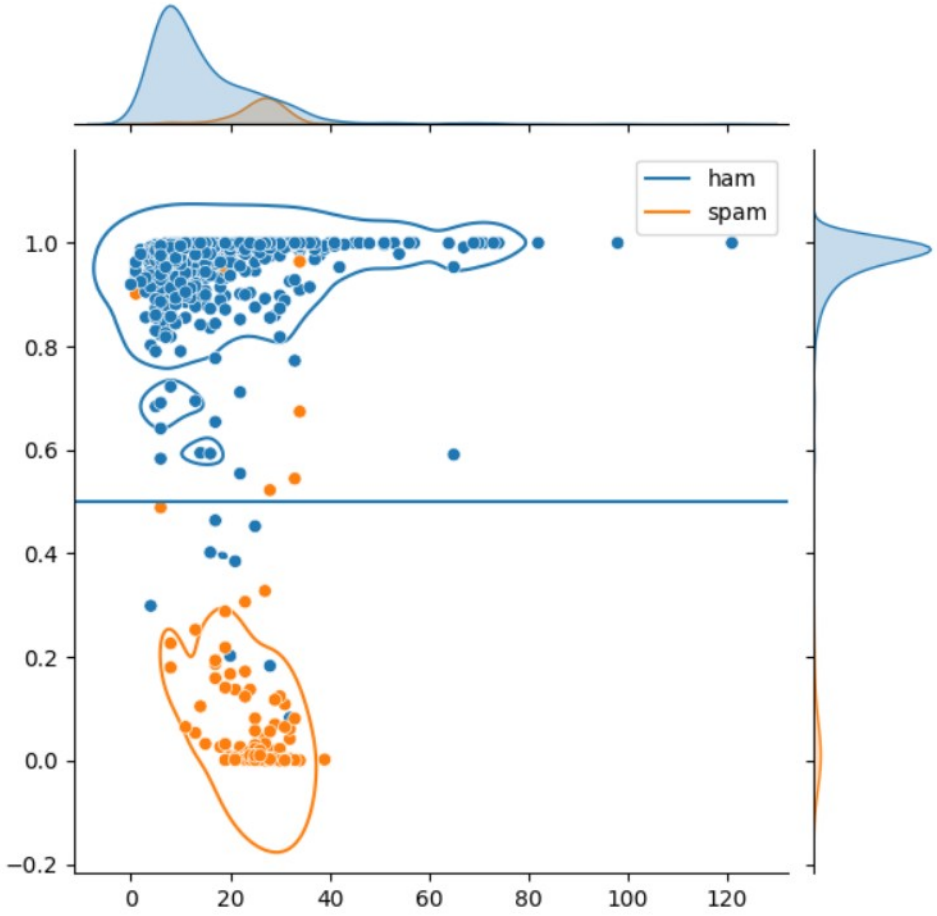

Figura 10: grafico errori rete neurale

4.6 Differenza e valutazioni tra i due modelli

L'utilizzo di una rete neurale è davvero giustificato per problemi come questo? Una giusta osservazione che sorge, è appunto la necessità di utilizzare una tecnologia tanto complessa e costosa come la rete neurale per la risoluzione di problemi di classificazione binaria.

Per ottenere una rapida risposta a questo quesito, nella presente sezione metteremo a confronto la risoluzione del problema sfruttando l'algoritmo di classificazione Naive Bayes. La libreria Scikit-Learn ci fornisce diverse implementazioni per l'algoritmo Naive Bayes: Naive Bayes - scikit-learn. Tra le tante, la scelta è comunque ricaduta sul **Multinomial Naive Bayes**.

4.6.1 Accuratezza del modello

L'accuratezza è una metrica per valutare i modelli di classificazione. A grandi linee, l'accuratezza è la parte della previsione che il nostro modello ha fatto in modo corretto. Formalmente, l'accuratezza è definita come segue:

$$AccuratezzaANN = \frac{TP + TN}{TP + FP + TN + FN} = 0.9874439461883409$$

$$AccuratezzaMNB = 0.7695067264573991$$

4.6.2 Precisione del modello

La precisione misura quale proporzione di identificatori positivi è effettivamente corretta, ed è definita come segue:

$$PrecisioneANN = \frac{TP}{TP + FP} = 0.9937888198757764$$

$$PrecisioneMNB = 0.9721115537848606$$

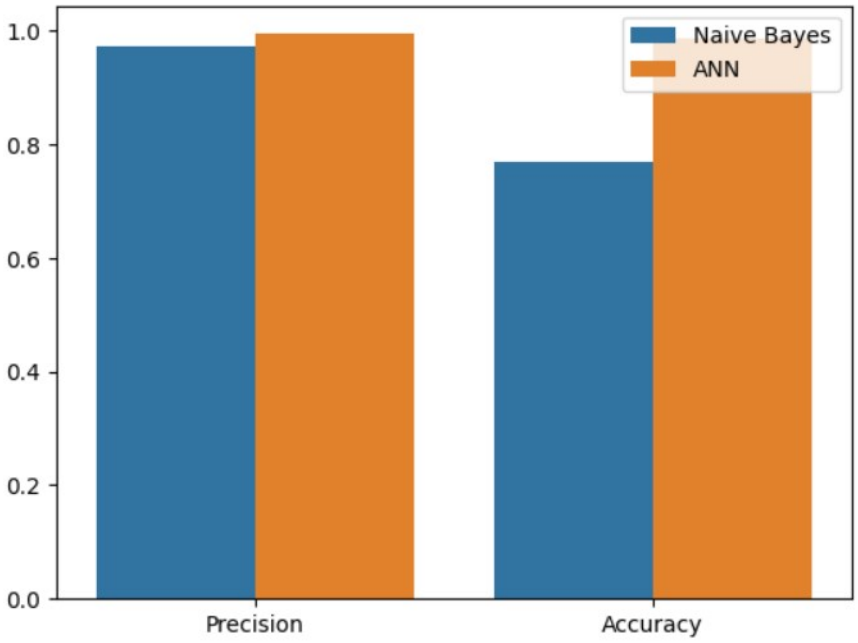

Figura 11: differenze

4.6.3 Matrice di confusione MNB

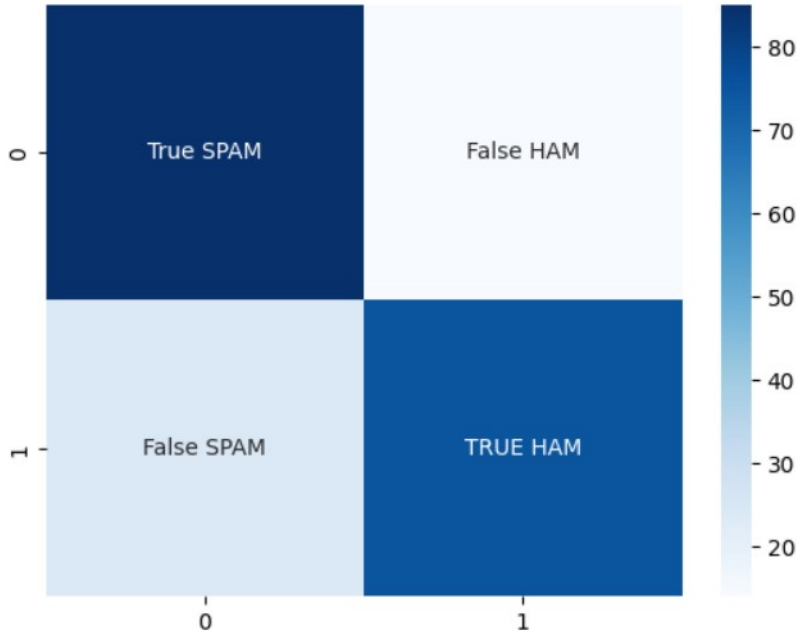

Figura 12: matrice di confusione naive bayes

Le matrici di confusione ottenute, ci riportano in entrambi i casi risultati abbastanza soddisfacenti, anche se leggermente migliori nel caso dell'uso della rete neurale.
Dunque, l'uso di un algoritmo con rete neurale risulta essere giustificato? ASSOLUTAMENTE NO (per questo progetto)! Infatti, il modello basato su Naive Bayes, ha riportato dei risultati comunque soddisfacenti, messi a confronto con quelli ottenuti dalla rete neurale. Chiaramente escludendo riscontri che potrebbero essere visibili solo in fase di distribuzione del modello in un ambiente reale. Quello che è davvero sorprendente è che i risultati, ottenuti dal classificatore della famiglia di Naive Bayes, sono quasi equivalenti a quelli ottenuti dalla rete neurale anche senza eseguire alcuna azione di bilanciamento dei dati e quindi senza modificare i pesi delle etichette.

4.7 Deployment

La fase di deployment ha l'obiettivo di mettere in funzione l'approccio definito e, quindi, renderlo usabile.

Il modello generato, precisamente, come dovrà essere reso disponibile? Quale sarà il grado di interazione con gli utenti? Ed in che modo gli utenti utilizzeranno il modello? In altri termini, questa fase vede il passaggio dall'ingegneria del machine learning all'ingegneria del software e all'ingegneria dell'usabilità!

Questo è ancora più evidente se consideriamo che questo modello non resterà nel suo stato in eterno e avrà bisogno di essere costantemente monitorato e manutenuto! Possiamo citare le prime due leggi di Lehman sull'evoluzione di sistemi software: (1) un sistema che non cambia è un sistema che diventerà presto inutile; (2) la complessità del sistema crescerà inesorabilmente nel tempo.

Il modello in questione può essere usato per vari scopi. Oggi questo tipo di modelli è usato tantissimo per il controllo delle mail spam. Questo però è solo il più comune tra gli usi. Possiamo pensare al controllo delle chiamate spam o meno, oppure per tutelare le carte di credito che ormai sono di uso quotidiano per tutti noi. Inoltre, il modello può essere usato per filtrare qualsiasi tipo di messaggio, in un ambiente in cui non ci interessa tenere conto di chi sia il mittente. Un caso reale potrebbe essere l'applicazione in ambienti di blogging o social, in cui si vuole effettuare una prima selezione di messaggi potenzialmente SPAM, i quali dovranno essere supervisionati da amministratori del sistema prima di essere resi pubblici

5 Miglioramenti

Durante la discussione del progetto sono sorte alcune problematiche, in questa sezione andremo ad evidenziare i miglioramenti effettuati.

- **1. L'uso della ANN**: Il primo punto che durante la discussione non è stato molto chiaro era quello riguardante il funzionamento e l'input della rete neurale usata.
  In merito a ciò bisogna chiarire prima di tutto cosa diamo in input alla rete neurale. Il tokenizzatore da noi addestrato nel momento in cui gli viene chiesto di tokenizzare un messaggio applica la tokenizzazione. In pratica, sostituisce a ogni token a lui noto individuato nel messaggio il numero corrispondente all'indice del vocabolario. Alla fine della fase di tokenizzazione avremo un array per ogni messaggio, nella quale, al posto delle parole abbiamo i token del nostro vocabolario, questi token rappresentano gli input che entreranno nella rete. In particolare, nella rete neurale che abbiamo utilizzato per questo progetto il primo livello è un livello di **Embedding**. Questo livello prende i messaggi tokenizzati e cerca un vettore di incorporamento per ogni indice di parola. I vettori aggiungono una dimensione all'array di output. Dopo il primo livello sono presenti due livelli intermedi che sono chiamati livelli di Dropout. I livelli Dropout impostano in modo casuale le unità di input su 0 con una frequenza di **rate** (passata come unico argomento) a ogni passaggio durante la fase di addestramento, che aiuta a prevenire l'overfitting. Gli input non impostati su 0 vengono aumentati di 1/(1 - rate) in modo tale che la somma di tutti gli input rimanga invariata. Possiamo dire che il secondo livello della nostra rete ha un rate pari a 0.2 mentre il terzo possiede un rate pari a 0.4. Successivamente, un livello **GlobalAveragePooling1D** restituisce un vettore di output a lunghezza fissa per ogni esempio calcolando la media sulla dimensione della sequenza. Ciò consente al modello di gestire input di lunghezza variabile, nel modo più semplice possibile. Infine l'ultimo strato è densamente connesso con un singolo nodo di output.

  Come nel nostro caso una rete neurale utilizza un algoritmo di **retropropagazione** dell'errore (backpropagation). Esso permette di modificare i pesi delle connessioni in modo tale che si minimizzi una certa funzione errore **E**. Tale funzione dipende dal vettore h-esimo di output restituito dalla rete, dato il vettore h-esimo di ingresso e dal vettore h-esimo di output che noi desideriamo (che fa parte del training set).

  Inoltre, il modello che abbiamo addestrato per questo progetto, può essere definito come "binario" o "a due classi", ciò indica che potrà fornirci risposte di tipo 0 e 1 o nello specifico per il nostro caso, "HAM" e "SPAM".

  Poiché si tratta di un problema di classificazione binaria e il modello emette una probabilità (uno strato di unità singola con un'attivazione sigmoidea).Infatti una delle scelte effettuate durante la fase di modeling, a grande impatto per le fasi successive, è quella di non arrotondare i valori di output della rete neurale al più vicino intero, dunque, sostituendo una più ovvia funzione di attivazione a passo unitario con una **funzione sigmoidea**, andando successivamente ad effettuare una arrotondamento con una funzione esterna alla rete.

  Questa scelta è stata particolarmente influenzata dal vantaggio di poter, in un ambiente in cui il modello non dovrà essere distribuito ma valutato, utilizzare le classificazioni per effettuare valutazioni, non solo binarie, le quali ci fornirebbero solo informazioni del tipo "Classificazione Sbagliata" o "Classificazione Corretta", piuttosto avere anche una misura di quanto il modello sia sicuro della sua scelta. Inoltre la funzione sigmoidea ci consente di evitare che le valutazioni del modello si vadano a concentrare in valori vicini a 0.5, relativi a casi di "estrema indecisione".

  Un grande aiuto per la costruzione del modello in questione è stato fornito dalle innumerevoli guide sull'utilizzo di TensorFlow, andando ad apportare piccole modifiche al codice fornito, basandosi su osservazioni fatte durante lo svolgimento delle diverse attività:
  Basic text classification - TensorFlow Core.
  Per maggiori informazioni su questo punto si rimanda alla sezione 4.4.

- **2. Data Cleaning**: Durante la discussione non era chiaro se in questo progetto avessimo fatto uso di tecniche di data imputation.
  In merito a ciò possiamo dire che vista la distribuzione sbilanciata delle etichette del dataset, si potrebbe pensare di voler svolgere una fase di data imputation, ma come vedremo successivamente, questa distribuzione NON sarà un problema per il nostro modello e quindi NON è stata applicata nessuna tecnica di dtata imputation.
  Inoltre non era chiaro il discorso sugli outlier. Per chiarire meglio possiamo dire che durante il progetto ci siamo chiesti se fosse necessario eliminare quei pochi messaggi che contengono un alto numero di parole e che quindi abbiamo considerato outlier.

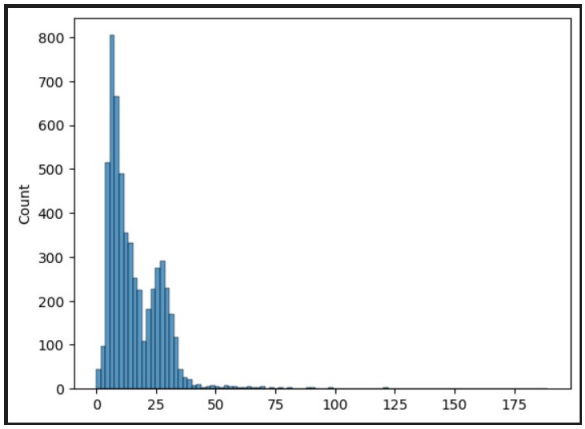

Figura 13: distribuzione per conteggio di parole

Lo svolgimento di questa fase si renderebbe necessaria solo nel momento in cui, nelle fasi successive all'addestramento, trovassimo una correlazione tra errori commessi dal modello e la lunghezza dei messaggi.
Per controllare se ci fosse questa correlazione abbiamo creato un grafico degli errori che ci permette di capire quando e come il nostro modello sbaglia. Il suddetto grafico è riportato di seguito.

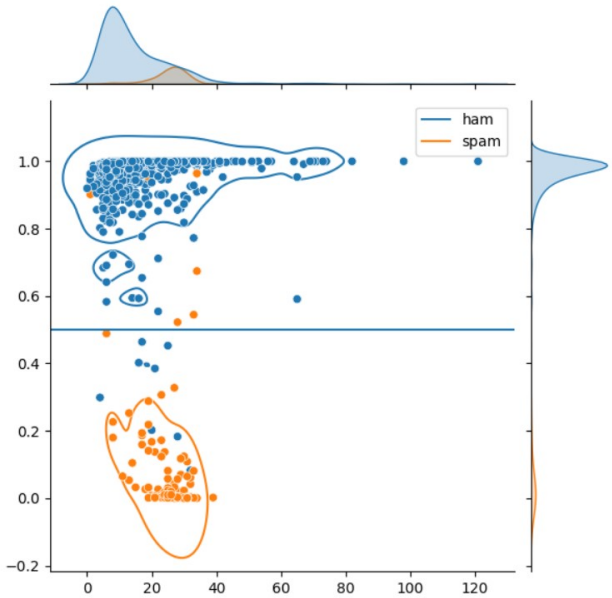

Figura 14: grafico errori rete neurale

Guardando il grafico ci accorgiamo che gli sbagli del modello non sono concentrati dove i messaggi sono più lunghi. Quindi, considerando che non abbiamo trovato nessuna correlazione tra la distribuzione per conteggio di parole e gli errori fatti dal nostro modello, abbiamo scelto di NON eseguire la pulizia per quanto riguarda gli outlier. Si precisa che potrebbe comunque esistere una caratteristica del dataset, non ancora individuata, associabile alla presenza di errori.
Un'attività che, invece, abbiamo eseguito per la fase di pulizia dei dati è quella di sostituire ogni cifra presente nei messaggi con il simbolo "#", in modo tale che possano essere ignorate in fase di tokenizzazione. Questa attività è stata necessaria per eliminare dati rumorosi dal nostro dataset, i numeri non sono fondamentali per il nostro scopo perché, anche se in un messaggio dato in input al modello ci sarà un numero, è molto difficile che questo numero sia riconosciuto dal modello e quindi già esistente nel dataset.

- **3. Data balancing**: Un altro dubbio che è sorto durante la presentazione del progetto riguardava la fase di bilanciamento di dati. Giustamente, vista la distribuzione di etichette del nostro dataset, la commissione si sarebbe aspettata l'applicazione di una tecnica di balancing.
  Per questo discorso possiamo dire che nonostante lo sbilanciamento dei dati, non abbiamo eseguito undersampling, perché rischiavamo di avere troppe poche istanze nella totalità del dataset e questo ci portava a delle predizioni meno esatte. Non abbiamo nemmeno aggiunto righe per quanto riguarda i messaggi considerati spam perchè non è stato possibile trovare messaggi adatti al nostro set di dati ed era troppo oneroso crearne di nuovi. Non potendo, quindi, aggiungere e/o togliere righe dal nostro dataset abbiamo deciso di seguire il suggerimento dato durante la presentazione del progetto. Seguendo il suggerimento, abbiamo dato un maggiore peso alla classe che risulta essere in minoranza, che nel nostro caso sarebbero i messaggi spam. La modalità di impostazione di questi pesi è stata derivata dalla documentazione di TensorFlow. La scelta dei pesi è pienamente imputabile a valutazioni empiriche effettuate reiterando l'attività di addestramento, apportando di volta in volta modifiche ai diversi pesi, fino al raggiungimento di un risultato soddisfacente. In particolare abbiamo dato peso 1 ai messaggi considerati "spam" e 0.2 ai messaggi "ham".

- **4. ANN vs Naive Bayes**: Durante la discussione del progetto sono sorti dubbi riguardanti la mancata motivazione dell'uso della rete neurale e il mancato confronto con un altri modelli di Machine Learning. Per tutte le info riguardanti il seguente punto si rimanda alle sezioni 4.4.2, 4.5 e 4.6.

6 Conclusioni

Seppure entrambi i modelli abbiano totalizzato un'accuratezza molto alta, andando a visualizzare in dettaglio i risultati, nonostante l'elevata accuratezza dei modelli nella classificazione di dati provenienti dal Dataset utilizzato per l'apprendimento, il comportamento presenta molti dubbi nel caso in cui li si utilizzi per predire l'etichetta di nuove stringhe (magari scritte a mano). Questo comportamento potrebbe essere una delle conseguenze della scarsa numerosità dei dati. Possiamo affermare con certezza, quindi, che entrambi i modelli soffrono di **overfitting** per il semplice motivo che se l'utente inserisce un messaggio spam non conforme ai messaggi spam presenti nel dataset i nostri modelli sbaglieranno le predizioni. Inoltre possiamo assumere che i modelli soffrono anche di **data leakage** proprio perché se lavorano sui dati presenti nel dataset abbiamo potuto vedere che si comportano in maniera quasi perfetta mentre in un ambiente reale sbaglieranno più del dovuto. Questo è influenzato dal fatto che anche solo analizzando manualmente i diversi messaggi all'interno del dataset si può riscontrare che questi sono tutti relativi a uno stesso contesto semantico, dunque in un ambiente reale i modelli funzioneranno correttamente solo se riceveranno messaggi scritti allo stesso modo di come sono scritti nel dataset

6.1 Possibili utilizzi del modello e osservazioni

Sulla base dei nostri risultati e della sperimentazione sul modello svolta mediante uno script Python realizzato ad-hoc, un modello di questo tipo può essere utilizzato in combinazione con uno svariato numero di ambienti. Un tipico esempio è il filtraggio dei commenti per le piattaforme blogging, o per un form di raccolta di questionari. Purtroppo però, le vulnerabilità del modello in questione sono diverse, infatti, un qualsiasi utente correrebbe in ogni caso il rischio di vedere i suoi innocenti messaggi classificati come SPAM... D'altronde questo è un fenomeno assai frequente anche nell'utilizzo di noti servizi di posta elettronica come **Gmail** o Outlook.

Nasce quindi spontanea una domanda, è più grave un **falso spam** o un **falso ham**? La domanda, come al solito potrebbe essere risolta con un più che generico "dipende", che aprirebbe la nostra mente verso una serie di specializzazioni applicabili al dominio di un problema di questo tipo: dipende dal messaggio? dipende dal mittente? dipende dai messaggi precedentemente scambiati con il mittente?

Per tutti i dettagli implementatiti si rimanda alla lettura del Jupite Notebook relativo al progetto.
Jupiter Notebook: https://github.com/SimoneC24/text-spam-checker/blob/main/REAMDE.ipynb