UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura

Corso di Laurea in Ingegneria Informatica Elettronica e delle Telecomunicazioni

Document embedding mediante Transformer per la predizione della bancarotta

Document embedding with Transformer for bankruptcy prediction

Relatore:

Chiar.mo Prof. Gianfranco Lombardo

Tesi di Laurea di:

Andrea Bertogalli

ANNO ACCADEMICO 2021-2022

Alla mia famiglia.

"L'entusiasmo è alla base di tutti i progressi."
Henry Ford

Indice

Elenco delle figure

Introduzione

Contesto dell'attività di tesi

La capacità di poter predire in anticipo una possibile bancarotta di un'azienda al giorno d'oggi è un elemento fondamentale nell'ambito del rischio del credito e si rivela fondamentale per figure quali creditori, investitori, azionisti e partner dell'azienda. Inoltre la crisi finanziaria del 2008 è un chiaro esempio di come il fallimento simultaneo di varie società abbia un impatto significativo sull'economia. Il tema della predizione della bancarotta è considerato un tema rilevante sin dal passato (Edward I. Altman, 1968) [1] ma oggi con l'avanzamento esponenziale della tecnologia abbiamo la possibilità di sviluppare modelli nettamente più avanzati, per questo il tema trattato è un tema di estrema rilevanza in ambito di ricerca.

Obbiettivo della tesi

L'obbiettivo dell'attività di tesi è quello di progettare e sviluppare un modello addestrabile end-to-end che abbia prestazioni migliori rispetto a modelli già esistenti per quanto riguarda il tema della predizione della bancarotta, in particolare che si basa sulla combinazione di una rete *LSTM multi-head* e di un *Transformer* in modo tale che la rete *LSTM* possa processare dati finanziari e il *Transformer* possa processare i documenti 10-K delle aziende statunitensi quotate in NYSE e NASDAQ dal 2000 al 2018. Più in particolare, per la parte di NLP, quindi quella di analisi dei documenti 10-K, è

stato sfruttato il fatto che i manager che stilano questi report su base annuale si sono rivelati "pigri", in quanto spesso riutilizzano il documento dell'anno precedente andando a modificare esclusivamente dettagli rilevanti al fine di rispettare i limiti della responsabilità fiduciaria (Cohen et al., 2020) [2]. Possiamo sintetizzare dunque l'architettura del modello proposto con il seguente schema a blocchi:

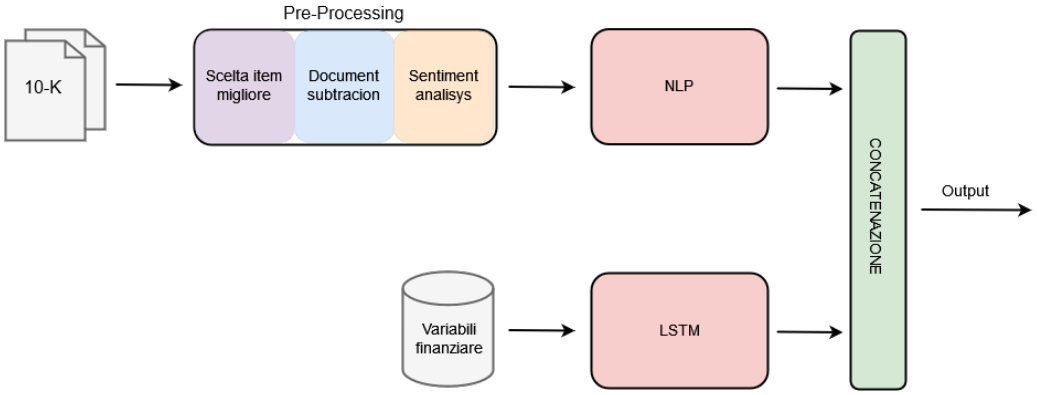

Figura 1: Schema generale del funzionamento del modello proposto.

Come possiamo notare dallo schema riportato sopra per la parte di *NLP* vengono effettuate alcune operazioni di pre-processing sui documenti *10-K*; successivamente i dati ottenuti sono processati dal modello di *NLP*. I due modelli, *LSTM* e *NLP* vengono infine combinati per produrre un modello unico, che produce un output che dipende sia dai *10-K* che dalle *variabili finanziarie*. Nei capitoli successivi verrà approfondito ogni blocco dello schema riportato.

Capitolo 1

Stato dell'arte

Sono numerose le pubblicazioni in ambito scientifico che trattano il tema della predizione della bancarotta, a partire dall'economista Altman che nel 1968 [1] individua la relazione tra sei variabili finanziarie e il rischio di fallimento. Formula infatti il punteggio Z-Score che permette una distinzione tra aziende sane dal punto di vista finanziario o sotto stress. Attualmente le pubblicazioni in questo ambito si basano, per la maggior parte, su strumenti di Machine Learning e Deep Learning.

Per quanto riguarda gli strumenti di *Machine Learning* sono stati fatti svariati tentativi, uno dei più significativi nel 2017 da parte di Barboza, Kimura e Altman [3], che hanno individuato altri sei indicatori che permettono un incremento in media del 10% rispetto agli indicatori individuati in precedenza. Inoltre esplorano diversi modelli di Ensemble Learning[1] individuando come tecnica migliore il Bagging[2] e come modello migliore il Random Forest[3].

[1]Per Ensemble Learning si intendono una serie di metodi d'insieme che usano modelli multipli per ottenere una migliore prestazione predittiva rispetto ai modelli da cui è costituito

[2]Il bagging è una tecnica di machine learning che rientra nella categoria dell'Apprendimento ensemble. Nel bagging più modelli dello stesso tipo vengono addestrati su dataset diversi, ciascuno ottenuto dal dataset iniziale tramite campionamento casuale con rimpiazzo

[3]Per Random Forest si intende un classificatore d'insieme ottenuto dall'aggregazione tramite bagging di alberi di decisione

Per quanto riguarda l'ambito del *Deep Learning*, svariati studi si occupano di definire la miglior tipologia di *rete neurale* per il task della predizione della bancarotta (Odom e Sharda, 1990 [4], Zhang et al., 1999 [5]) ed è stato dimostrato da Vochozka et al., 2020 [6] che le reti LSTM sono la tipologia di reti che performano meglio su questa tipologia di task. Proprio da questo deriva infatti la scelta di utilizzare questa tipologia di rete per il modello proposto.

1.1 Reti ricorrenti e LSTM

Le reti neurali ricorrenti (**R**ecurrent **N**eural **N**etwork) sono state introdotte con lo scopo di interpretare un dato facendo riferimento anche ai dati precedenti; per esemplificare il concetto basta pensare al fatto che leggendo questa tesi per comprendere le parole occorre aver compreso quelle precedenti, cosa che le reti classiche non sono in grado di fare. A questo fine le *RNN* utilizzano dei cicli, ovvero i valori di uscita di un layer di livello superiore (più vicino all'uscita) vengono utilizzati come ingresso per un layer di livello inferiore (più vicino all'ingresso). Questa retroazione, in sostanza, fa sì che l'uscita $O(t)$ (ovvero all'istante t) sia funzione dell'ingresso $X(t)$ e, in parte, del valore $V(t)$ che, per effetto del ciclo, corrisponde all'uscita precedente $O(t-1)$. Ma l'uscita $O(t-1)$, a sua volta, dipendeva dal valore dell'ingresso $X(t-1)$ e di $V(t-1)$ che, sempre per effetto del ciclo, corrispondeva all'uscita ancora precedente $O(t-2)$ e così via. Per questo motivo, le *RNN* si prestano all'elaborazione di sequenze sia temporali che di altro genere. In figura 1.1 viene riportato uno schema dove, a sinistra, vediamo una rete ricorrente composta da un singolo neurone e, a destra, vediamo il suo comportamento a diversi istanti di tempo.

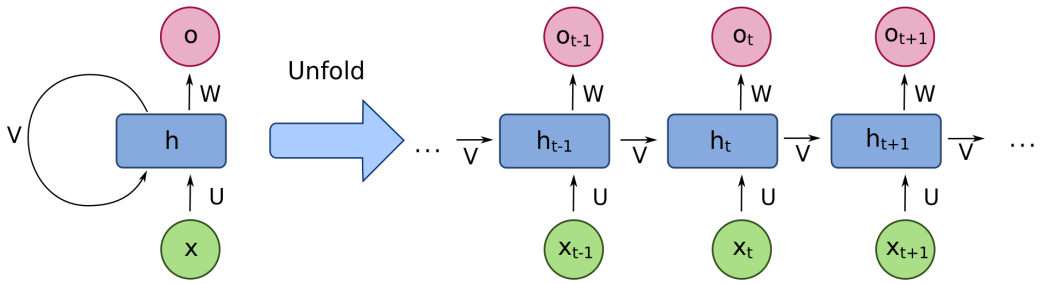

Figura 1.1: Schema che rappresenta una RNN con un singolo neurone.

Un problema tipico delle reti ricorrenti è quello del cosiddetto *vanishing o exploding gradient*. Il *vanishing gradient* si verifica quando, durante l'algoritmo di *Backpropagation*[4], utilizziamo funzioni di attivazione come la funzione sigmoide o la tangente iperbolica, che sono funzioni con gradiente che varia tra 0 e 1. Di conseguenza, la moltiplicazione a catena dei gradienti durante l'algoritmo tende a far diminuire drasticamente la capacità di aggiornamento dei pesi. Il problema di *exploding gradient* si verifica, invece, quando utilizziamo funzioni di attivazione come la ReLU o la funzione lineare; in questo caso i valori dei gradienti possono essere maggiori di 1 e, di conseguenza, abbiamo una crescita incontrollata del valore complessivo.

Per ovviare a questi due problemi, che si riscontrano qualora si elaborino lunghe sequenze di dati, sono state introdotte le reti *LSTM* (**L**ong **S**hort-**T**erm **M**emory) (Hochreiter e Schmidhuber, 1997 [7]). Le reti *LSTM* utilizzano neuroni più complessi rispetto alle reti ricorrenti classiche e permettono, oltre che a limitare il problema del *vanishing o exploding gradient*, di elaborare sequenze più lunghe. Nella figura sottostante viene riportata la differenza tra un neurone di una *RNN* e un neurone di una rete *LSTM*:

[4] Algoritmo riguardante l'aggiornamento dei pesi dei vari neuroni in una rete neurale.

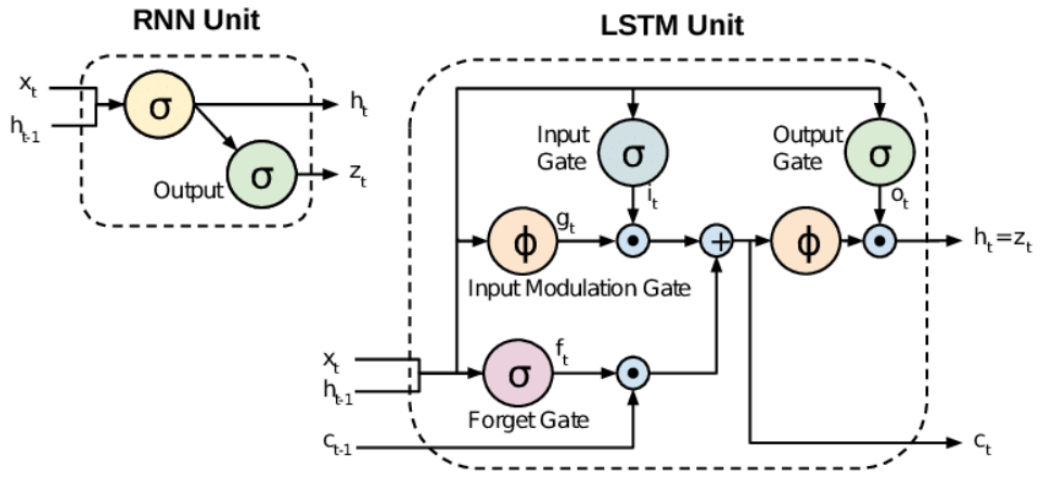

Figura 1.2: Neurone di una RNN classica comparato ad un neurone di una rete LSTM.

Mentre il neurone ricorrente gestisce semplicemente un'uscita supplementare $h(t)$ per lo stato attuale e un ingresso $h(t-1)$ per lo stato precedente, oltre che due funzioni di attivazione differenti per lo stato h e l'uscita z, il neurone LSTM ha internamente diverse porte (o gate), che gli consentono di decidere in autonomia (durante la fase di addestramento) cosa memorizzare o dimenticare, se e come combinare l'ingresso con lo stato interno, se e come restituire l'uscita. I gate che l'unità *LSTM* usa sono:

- Il **forget gate** si occupa di decidere se l'informazione in ingresso deve essere buttata o mantenuta mediante le informazioni dell'input corrente $x(t)$ e dell'uscita precedente in retroazione $h(t-1)$. A queste informazioni è applicata una funzione di attivazione sigmoidale, che restituirà un'uscita compresa tra 0 ed 1. Questa uscita verrà moltiplicata al valore dello stato precedente $c(t-1)$. Quindi, se la funzione sigmoide restituisce un valore prossimo a 0, lo stato precedente tenderà ad azzerarsi (viene dimenticato), mentre, se restituisce un valore prossimo ad 1, lo stato precedente tenderà a restare uguale (viene memorizzato).

- L'**input gate**, in modo analogo al forget gate, decide se i valori in ingresso $x(t)$ e $h(t-1)$ possono essere elaborati insieme allo stato precedente $c(t-1)$.
- Infine, l'**output gate**, in maniera analoga agli altri due gate, sfruttando sempre i valori in ingresso $x(t)$ e $h(t-1)$, decide se lo stato attuale $c(t)$ può essere presentato in uscita e diventare l'uscita z(t), che corrisponde anche al prossimo ingresso in retroazione $h(t)$.

Attraverso i vari stati, la rete *LSTM* è in grado di sviluppare il concetto di memoria a lungo termine e, di conseguenza, si è scelto di utilizzare questa rete per il task di predizione della bancarotta.

1.2 NLP

Le tecnologie considerate finora non prevedono l'interpretazione di dati testuali, ma solo di dati numerici come le variabili finanziarie. Per l'elaborazione dei 10-K dobbiamo fare riferimento all'ambito del *NLP*[5] (**N**atural **L**anguage **P**rocessing). Il mondo del natural language processing è molto vasto e vi sono vari task associati ad esso come, per esempio, la *Sentiment Anlalisys*, che mira ad attribuire una polarità (positivo, negativo, neutro) ad un testo. Distinguiamo, inoltre, vari algoritmi, che permettono di interpretare parole o documenti e generare vettori numerici (*embedding*), che ne esprimono il significato. Tra questi riconosciamo: *Tf-Idf* (Salton e Buckley, 1988 [8]), che è basato sulla frequenza con cui appaiono le parole in un documento; *Word2Vec* (Mikolov et al., 2013 [9]), che permette di creare embedding di singole parole utilizzando una particolare rete neurale; *Doc2Vec*, che è un derivato di *Word2Vec* e permettere di generare embedding di interi documenti. Se pur questi strumenti siano ancora considerati efficaci, attualmente lo stato dell'arte, per quanto riguarda il natural language processing, è

[5]Per Natural Language Processing o elaborazione del linguaggio naturale si intendono algoritmi di intelligenza artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale.

rappresentato dai *Transformers*. Il primo modello considerabile tale è stato introdotto nel 2017 facendo riferimento all'architettura proposta in *Attention is all you need*, Vaswani et al. [10]. I *Transformers* sono reti neurali che si basano su sofisticati meccanismi di attenzione e sono stati introdotti per risolvere il problema di *Long-term dependecy* che affligge le reti ricorrenti, ovvero quando si ha una lunga sequenza di parole le RNN tendono a "dimenticare" le prime parti della sequenza. I Transformers a differenza delle RNN hanno una memoria a lungo termine e riescono ad interpretare meglio il contesto dell'intera frase, anche se questa è di una certa lunghezza. Nella figura sottostante viene riportata l'architettura generale di un Transformer:

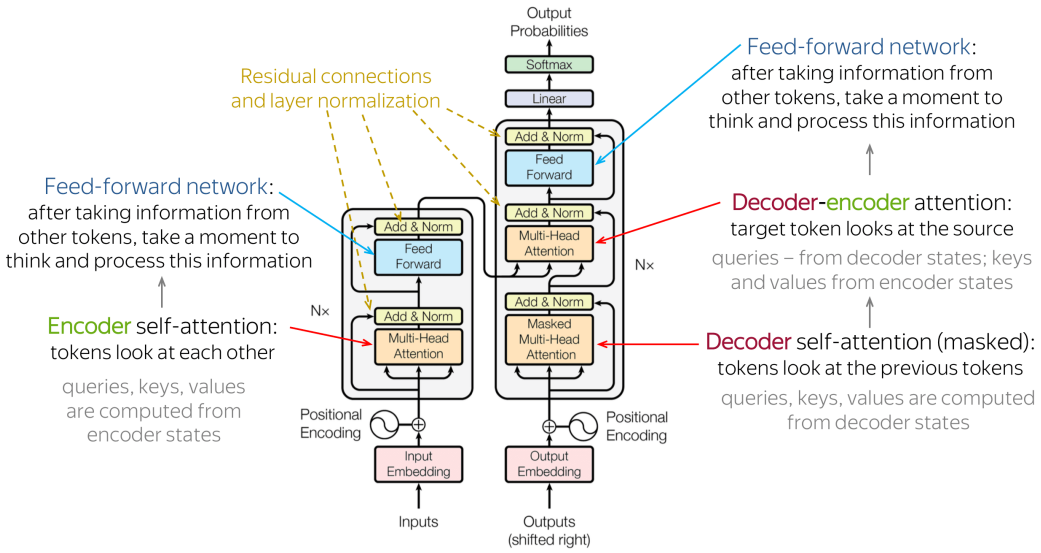

Figura 1.3: Architettura generale di un Transformer, *"Attention is all you need, 2017"* [10].

Come già specificato in precedenza, i Transformers si basano su meccanismi di attenzione. Con attenzione intendiamo che il modello dà più importanza (presta maggior attenzione) a certe parti di una frase, proprio come un essere umano quando legge un testo. Il meccanismo di attenzione (attention-mechanism) riconosce, inoltre, la semantica di una frase, fornendola al Decoder sotto forma di keywords. Per ogni parola letta dal LSTM

(*Encoder*), l'attention-mechanism prende in considerazione altri input allo stesso tempo, decidendo quali siano importanti e attribuendo loro un peso differente; il *Decorer* è quindi incaricato di prendere le parole corredate di un peso (weight). Facendo riferimento alla figura 1.3 abbiamo l'Encoder sulla sinistra e il Decoder sulla destra, entrambi modulari e formati da uno stack di livelli Feed Forward e Multi-Head Attention.

1.2.1 BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) è un modello sviluppato e pre-addestrato da Google (2018) [11] su vasti insiemi di documenti (tra cui l'intero wikipedia), che poi necessita solo di essere addestrato dall'utilizzatore su un task più specifico. Questo permette anche agli utilizzatori di disporre in modo indiretto di tutta la potenza di calcolo che Google ha impiegato nell'addestramento principale di BERT.

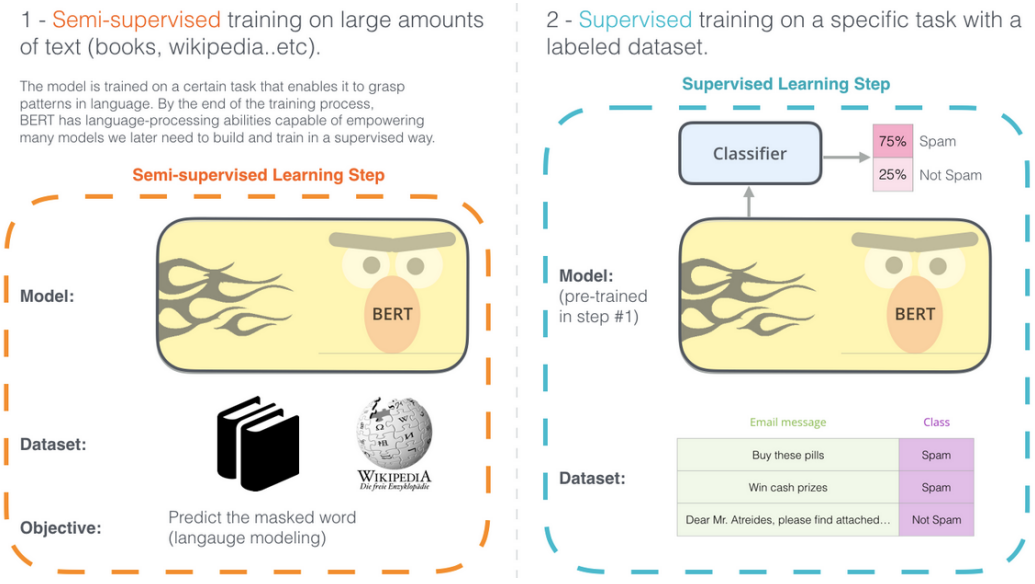

Figura 1.4: L'addestramento del modello BERT nei suoi 2 step Pre-train, fine-tuning (Jay Alammar, 2018) [12].

La caratteristica principale di BERT è quella di essere in grado di interpretare in maniera sorprendentemente efficace il contesto di una frase, anche di lunghezza elevata, andando a simulare in maniera accurata il modo di interpretare i discorsi di un essere umano. Per comprendere meglio che tipologia di associazioni riesce a fare BERT in base al contesto, in figura 1.5 viene riportata la rappresentazione grafica del meccanismo di attenzione che utilizza.

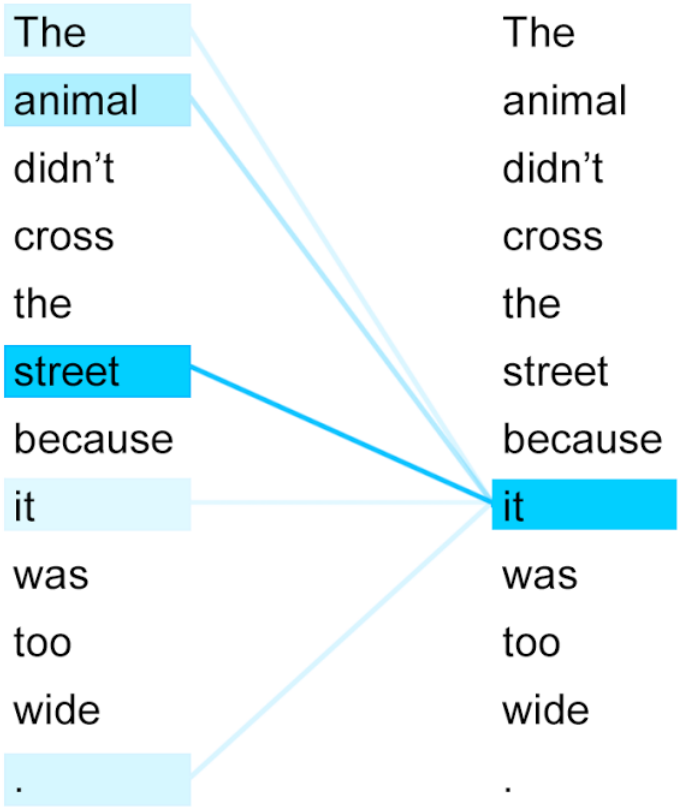

Figura 1.5: Rappresentazione grafica di come BERT utilizza il meccanismo di Self-attention per interpretare il contesto della frase, Prateek Joshi, 2019 [13].

Nella frase *"The animal didn't cross the street because* ***it*** *was too wide"*, è semplice, per noi umani, capire che *it* si riferisce alla strada, ma per un mo-

dello di intelligenza artificiale non è affatto banale. BERT però, mediante il meccanismo di *Self-attention*, riesce ad associare con probabilità maggiore *it* a *street* basandosi sul contesto. Abbiamo già introdotto, parlando dei Transformers in generale, cosa si intende per *Attention mechanism*, infatti è proprio grazie ad un Transformer bidirezionale, che è al centro dell'architettura di BERT, che è possibile l'utilizzo del meccanismo di Self-attention.

Ma come riesce BERT ad "imparare" il contesto delle parole? BERT "impara" il contesto di una parola, data in input una frase, utilizzando quello che si chiama un ***M**asked **L**anguage **M**odel*: il 15% delle parole nella frase viene mascherato mediante l'utilizzo del token [MASK] e BERT ha il compito di restituire in output la stessa frase in input; in questo modo riesce ad "imparare" il contesto delle parole.

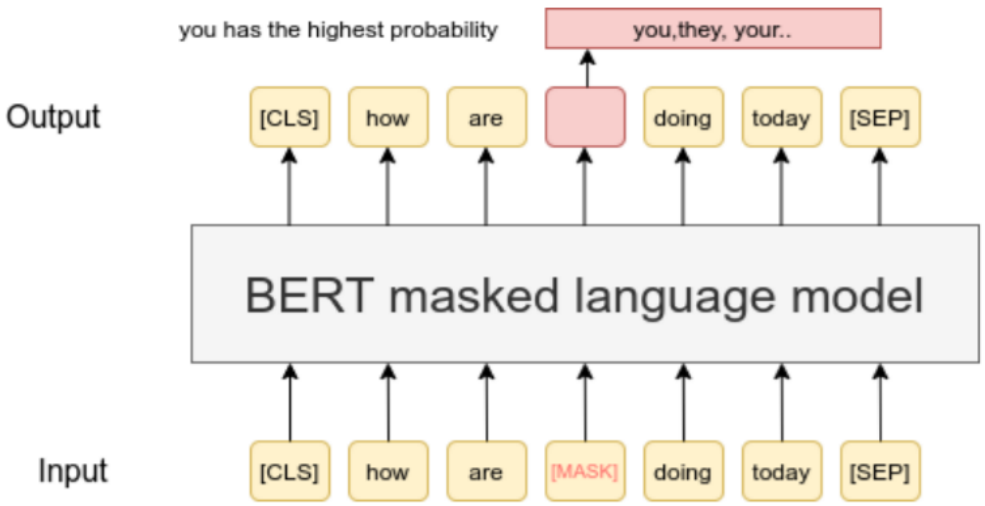

Figura 1.6: Esempio dell'utilizzo del Masked Language Model da parte di BERT, Abhilash Jain et al. 2020 [14].

Come notiamo in figura 1.6 sono presenti altri due token speciali (aggiunti durante il processo di *tokenization* necessario prima di dare in input i dati a BERT): il token [CLS] e il token [SEP]. Il token [SEP] si occupa semplicemente di separare le frasi, mentre il token [CLS] è di estrema importanza

in quanto, una volta che BERT ha processato un testo, l'embedding del token [CLS] corrisponde all'embedding del'intero testo processato, che risulta fondamentale in task di classificazione di documenti.

I task per cui è utilizzato BERT sono svariati, tra i principali riconosciamo la *Next Sentence Prediction*, la *Sentiment Analisys* e la *Document Classification.*

DistilBERT

Nonostante BERT sia pre-addestrato, risulta comunque oneroso dal punto di vista computazionale. Per questo, viene introdotto nel 2019 DistilBERT (*DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, Sanh et al. [15]), che permette di ottenere risultati simili a BERT, richiedendo molta meno potenza di calcolo. Infatti, permette di mantenere il 97% delle potenzialità di BERT riducendone del 40% la complessità. DistilBERT è frutto di quello che viene definito un addestramento di tipo *Teacher-Student*, particolare tecnica che rientra nel campo della *Knowledge-Distillation* (Hinton et al., 2015) [16], che consiste nell'addestrare un modello (Student) in modo tale da emulare i risultati in output e minimizzare la stessa funzione di loss di un altro modello già addestrato (Teacher). DistilBERT, in questo caso, rappresenta il modello student e BERT il modello teacher.

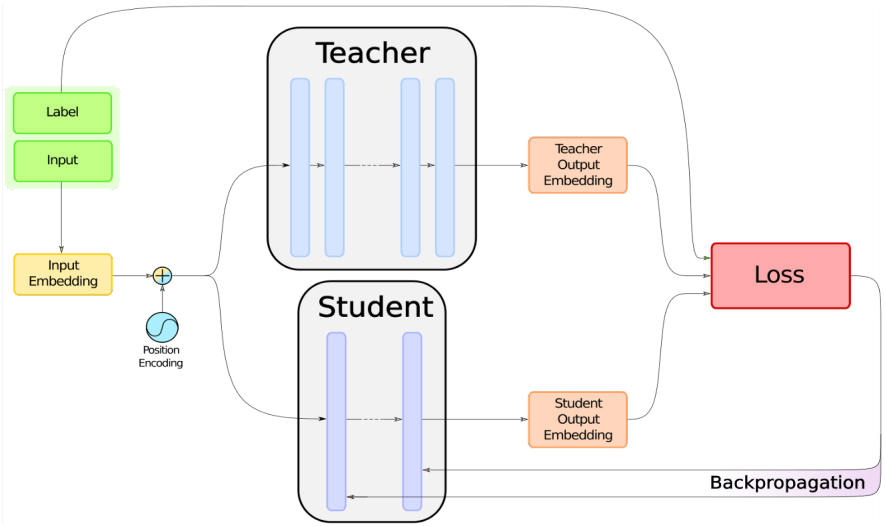

Figura 1.7: Esempio dell'utilizzo della tecnica di Knowledge-Distillation mediante un addestramento di tipo Teacher-Student [17].

Esattamente come per BERT anche con DistilBERT è possibile estrarre i *sentence-embeddings*. Il processo è molto simile: anche per DistilBERT si segue un'operazione di *Tokenization* che divide ogni singolo token nella frase, aggiunge i token speciali [SEP] e [CLS] e sostituisce ad ogni token il proprio id. DistilBERT genererà un vettore di lunghezza 768 per ogni token e il vettore associato al token [CLS] rappresenta l'embedding dell'intero input.

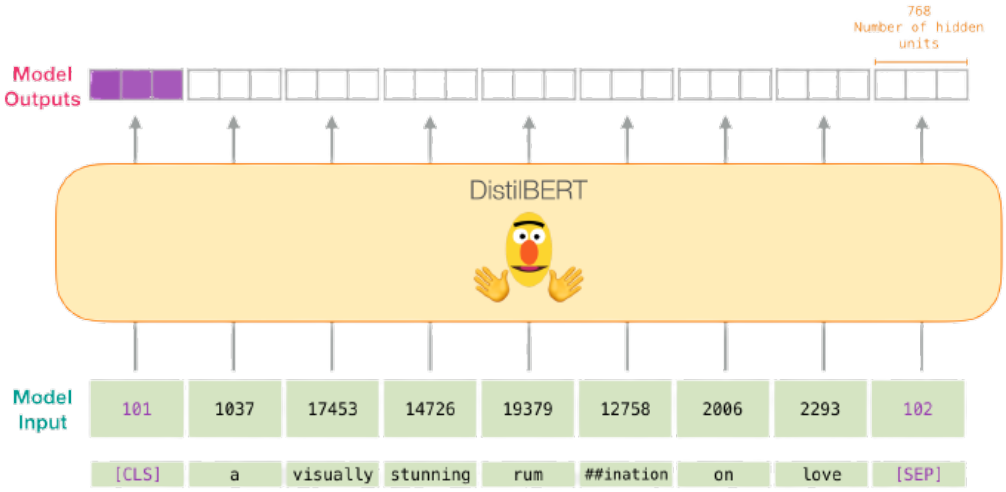

Figura 1.8: Rappresentazione del processo di estrazione degli embeddings da DistilBERT, Jay Alammar 2019 [18].

FinBERT

FinBERT (Dogu Araci, 2019 [19]) è una versione di BERT che è stata addestrata su un vastissimo insieme di documenti finanziari (FINWEB, Yahoo-Finance e RedditFinanceQA). È dimostrato dallo stesso autore del modello che FinBERT, per quanto riguarda l'analisi di documenti finanziari, supera di gran lunga ogni modello di Machine Learning anche sfruttando dataset di dimensioni ridotte.

1.2.2 Doc2Vec

Doc2Vec non rappresenta lo stato dell'arte per quanto riguarda il *Document embedding*, ma è da considerarsi un valido confronto con le tecniche più avanzate, tra le quali, per esempio, BERT. Doc2Vec è stato presentato in *"Distributed Representations of Sentences and Documents"* (Mikolov e Le, 2014) [20] ed è una tecnica semplice che, come si può dedurre dal nome, è fortemente legata a Word2Vec (Mikolov et al., 2013 [9]), infatti, per comprendere adeguatamente il funzionamento di Doc2Vec occorre conoscere

il funzionamento di Word2Vec. Quando si vuole costruire un modello che interpreti parole, usare metodi come il *One-Hot Encoding* può essere una strada accettabile, però utilizzare questa tipologia di tecniche fa perdere il significato alla parola e non può essere messa in relazione con altre parole. Se, per esempio, codifichiamo la parola "Paris" come id_4, la parola "France" come id_6 e la parola "Power" come id_8, la parola "France" avrà la stessa relazione sia con "Paris" che con "Power". Quello che noi vorremmo è che la parola "Paris" sia più correlata alla parola "France" di quanto lo sia la parola "Power": Word2Vec è in grado di catturare questa tipologia di relazioni.

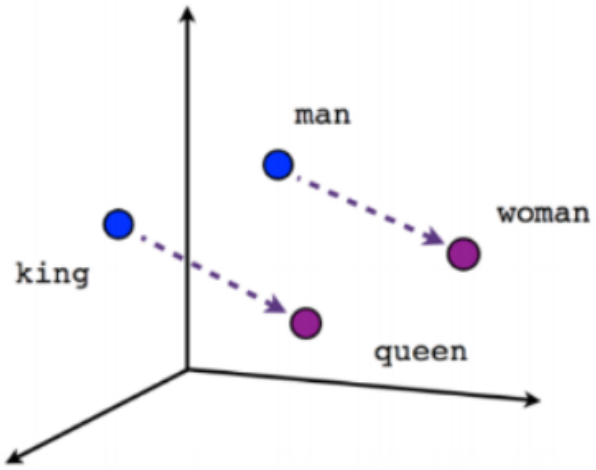

Figura 1.9: Con Word2Vec tra la parola "King" e la parola "Queen" abbiamo una relazione molto simile alla relazione tra la parola "Man" e la parola "Woman".

Word2Vec basa il suo funzionamento su due modelli, ***C**ontinuous **B**ag-**O**f-**W**ords (CBOW)* e *Skip-Gram*. CBOW crea una *sliding-window*, ovvero un intervallo di parole attorno alla parola che si vuole imparare; questo intervallo rappresenta il contesto della suddetta parola. Come già precisato precedentemente, dopo il training avremo che parole simili saranno rappresentate da vettori che sono vicini per qualche misura di distanza. CBOW può essere quindi usato per predire una parola in base al contesto. Skip-Gram è un algoritmo con il compito opposto a CBOW, infatti, data una parola,

usiamo Skip-Gram per predire il contesto di questa parola, ovvero le parole che la circondano.

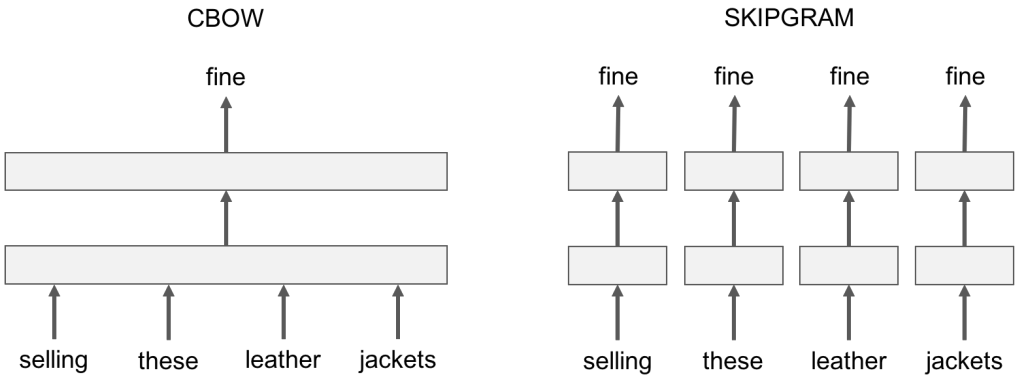

Figura 1.10: Differenza di funzionamento tra CBOW e SkipGram sull'esempio *"I am selling these fine leather jakets"*.

Dopo aver compreso a grandi linee il funzionamento di Word2Vec, è possibile introdurre Doc2Vec. Come già specificato, il compito di Doc2Vec è quello di generare una rappresentazione vettoriale di un intero documento. Dato che i documenti non seguono un ordine logico come le parole, occorre introdurre alcuni cambiamenti. L'idea proposta da Mikolov e Le è quella di utilizzare Word2Vec aggiungendo un vettore: il *paragraph-id*.

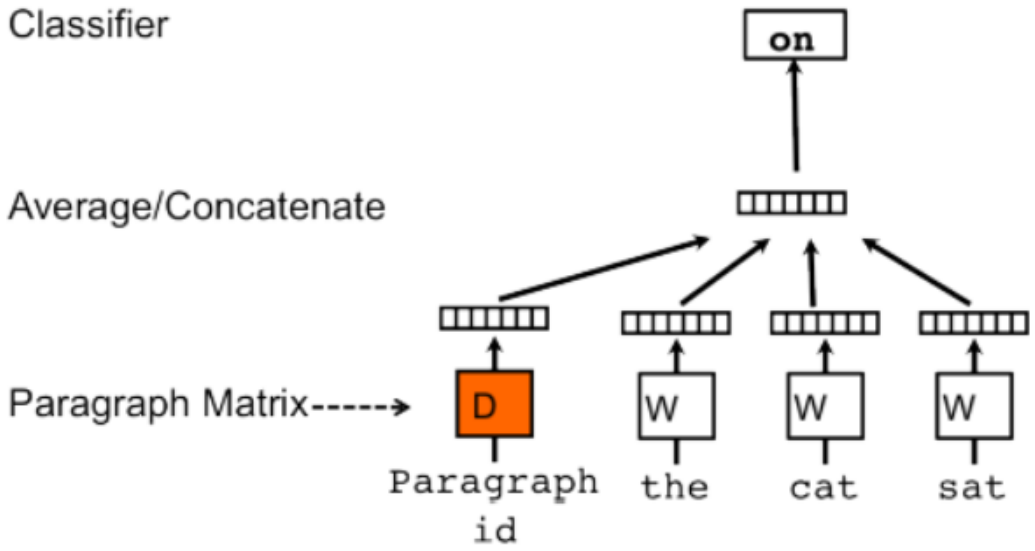

Figura 1.11: Rappresentazione del modello PV-DM usato da Doc2Vec.

Il modello in figura 1.11 è simile a CBOW, con la differenza che per predire la parola "mancante", oltre che alle parole di contesto, utilizziamo anche il paragraph-id, che è univoco su tutto il documento. Quando andiamo, quindi, ad addestrare il modello, il vettore paragraph-id assume una rappresentazione del documento stesso. Il modello in figura prende il nome di *Distributed Memory version of Paragraph Vector (PV-DM)*. Mentre un *word embedding vector* rappresenta il concetto di una parola, il *document embedding vector* rappresenta il concetto del documento. Come in Word2Vec avevamo un modello opposto al CBOW, lo Skip-Gram, anche in Doc2Vec è presente un altro modello che funziona all'opposto di PV-DM, il *Words version of Paragraph Vector (PV-DBOW)*.

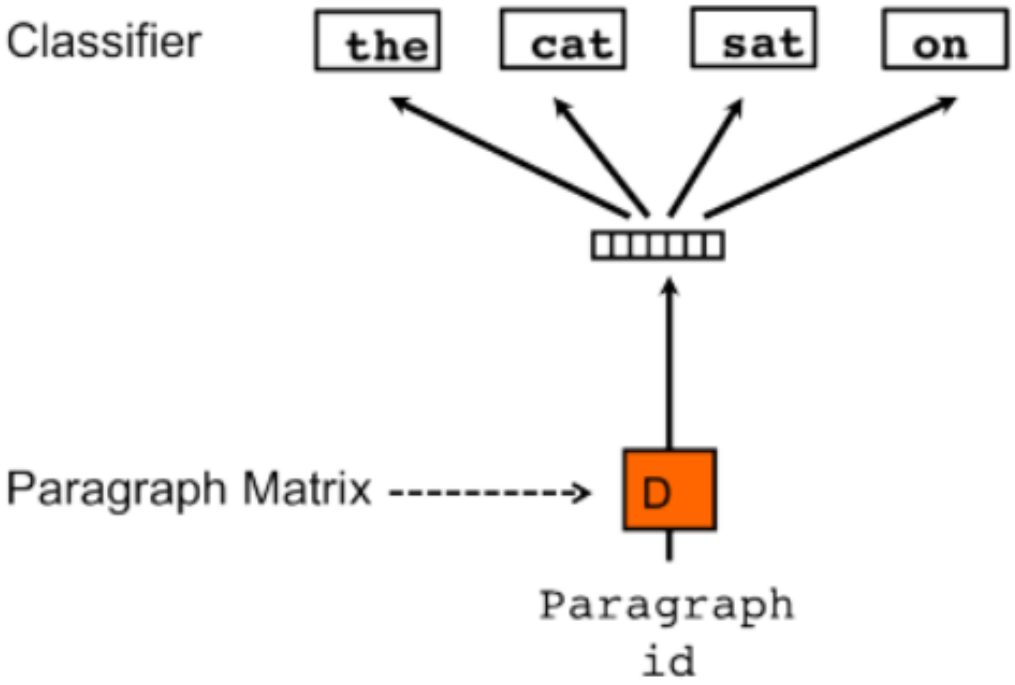

Figura 1.12: Rappresentazione del modello PV-DBOW usato da Doc2Vec.

1.3 Tecniche di unione di vettori

Durante la progettazione del modello di NLP si sono dovute utilizzare alcune tecniche di accorpamento di vettori, infatti si è dovuto accorpare più vettori di embedding, in modo tale da ottenere un embedding unico. A questo proposito occorre trovare una funzione che abbia, come input, un numero variabile di vettori e, in output, un vettore con dimensioni uguali ai vettori di input.

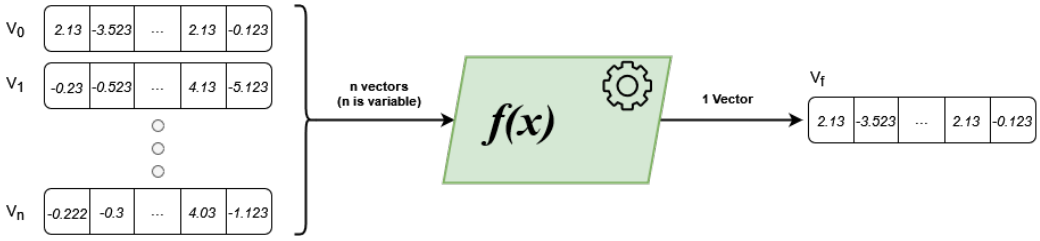

Figura 1.13: In figura è riportato lo schema del risultato che si vuole ottenere.

Funzioni di questo tipo sono state considerate in *"node2vec: Scalable Feature Learning for Networks"* (Aditya Grover e Jure Leskovec, 2016) [21], nel quale viene presentato l'algoritmo Node2Vec. Nell'articolo viene proposto un confronto tra varie tecniche per l'elaborazione di embedding di nodi in un grafo. Le tecniche tentate da Grover e Leskovec sono riportate in figura 1.14.

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|f(u) \cdot f(v)\|_{\bar{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|f(u) \cdot f(v)\|_{\bar{2}i} = \|f_i(u) - f_i(v)\|^2$

Figura 1.14: In figura sono riportate le tecniche usate da Aditya Grover e Jure Leskovec nello sviluppo dell'algoritmo Node2Vec.

1.3.1 Prodotto di Hadamard

Definizione 1.1. *In matematica, il prodotto di Hadamard, anche conosciuto come prodotto elemento per elemento o prodotto di Schur, è un operatore binario che, dati due vettori, produce un altro vettore con le stesse dimensioni degli operandi, dove ogni elemento i è il prodotto degli i-esimi elementi nei due vettori originali.*

$$(V_1 \boxdot V_2)_\mathrm{i} = (V_1)_\mathrm{i}(V_2)_\mathrm{i} \tag{1.1}$$

L'idea per ridurre un numero variabile di vettori in un unico vettore con le stesse dimensioni degli operandi è, quindi, quella di applicare iterativamente il prodotto di Hadamard su tutti i vettori:

$$V_{\mathrm{f}} = \prod_{i=0}^{n} V_i = V_0 \boxdot V_1 \boxdot ... \boxdot V_n \tag{1.2}$$

dove $\boxdot$ è il prodotto di hadamard tra due vettori.

1.3.2 Media tra vettori

Tecnica alternativa al prodotto di Hadamard è utilizzare la media elemento per elemento dei vettori (*element-wise mean*). Questa tecnica è più semplice, ma permette anch'essa di andare a ridurre n vettori di dimensione m in 1 vettore di dimensione m, proprio come il prodotto di Hadamard. Facendo riferimento ai chunks possiamo formalizzare l'operazione svolta nel seguente modo:

$$V_f = \frac{1}{n} \sum_{i=0}^{n} V_i \tag{1.3}$$

Ricordiamo che la somma tra vettori è una somma elemento per elemento definita come:

$$\begin{bmatrix} v_0 \\ v_1 \\ ... \\ v_m \end{bmatrix} + \begin{bmatrix} u_0 \\ u_1 \\ ... \\ u_m \end{bmatrix} = \begin{bmatrix} v_0 + u_0 \\ v_1 + u_1 \\ ... \\ v_m + u_m \end{bmatrix} \tag{1.4}$$

Quindi, sommando tutti gli n vettori e dividendo per n, si ottiene un unico vettore di dimensione m.

Capitolo 2

Architettura software realizzata

Come già specificato nel capitolo introduttivo, il modello proposto per l'attività di tesi è un modello composto da due parti principali: una parte che si occupa di variabili finanziarie e una parte che si occupa dell'analisi dei documenti 10-K (NLP). Il task di previsione della bancarotta di un'azienda è stato interpretato come un task di *classificazione binaria*[1], infatti possiamo distinguere solamente due classi da predire:

- Classe 0, azienda sana, non fallita.
- Classe 1, azienda non sana, fallita.

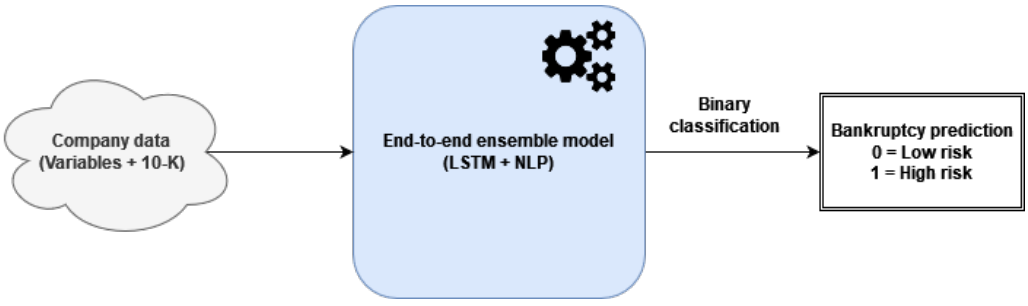

Figura 2.1: Rappresentazione di come il modello effettua il task di binary classidication.

[1]La classificazione binaria è un'attività di apprendimento automatico con supervisione che viene usata per stabilire a quale di due classi (categorie) appartiene un'istanza di dati.

2.1 Architettura del modello

In figura 2.2 viene riportata l'architettura proposta per il modello progettato per il task di predizione della bancarotta. Si tratta di un modello che è addestrabile end-to-end, questo significa che basta un solo addestramento del modello complessivo (non è necessario addestrare singolarmente vari modelli) e dopodiché il modello è pronto per essere utilizzato.

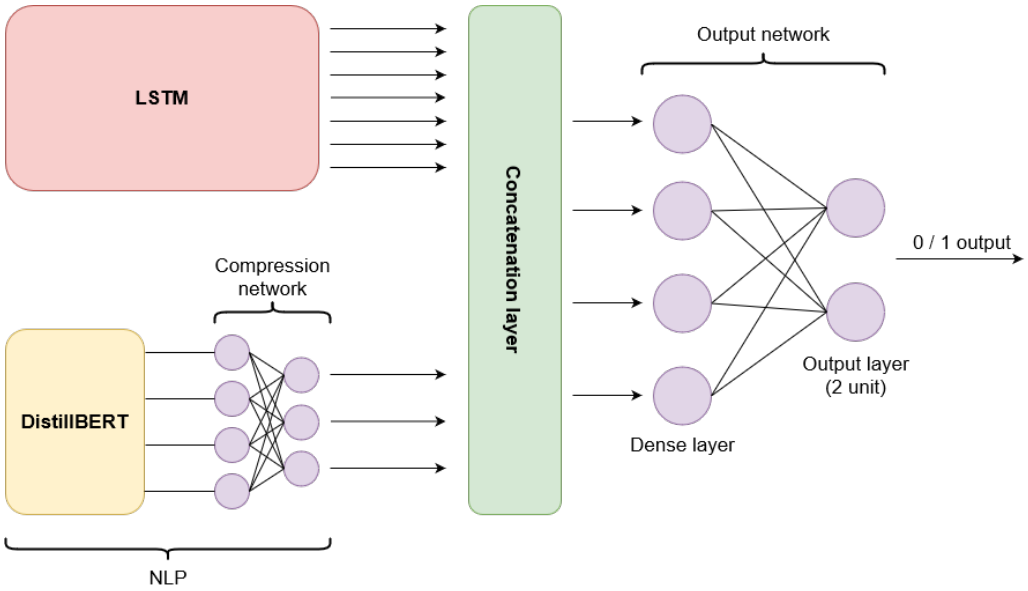

Figura 2.2: Rappresentazione semplificata dell'architettura del modello proposto.

Come notiamo, in figura possiamo distinguere la rete LSTM e il modello di NLP. La rete LSTM, che si occupa dell'elaborazione delle variabili finanziarie, è una rete composta da un input layer e da un layer LSTM da 18 teste, pari al numero di features in ingresso. Il modello di NLP ha una struttura più complessa di quella della LSTM, in quanto, oltre che ha tutta la parte di preprocessing dei 10-K, è formato da un modello di NLP come per esempio DistilBERT, che ha il compito di generare degli embedding, e da una rete di compressione che serve per elaborare ulteriormente gli embedding prima di essere dati in input alla rete di output. Infine le due parti principali (LSTM,

NLP) vengono concatenate mediante il layer *CONCATENATE* ad una rete neurale composta da 2 layers: un layer *DENSE* da 20 neuroni e funzione di attivazione *ReLU* e un output layer *DENSE* composto da 2 neuroni con funzione di attivazione *Softmax*. Mentre la parte di analisi delle variabili finanziarie non è particolarmente complessa, in quanto costituita da una sola rete LSTM, la parte di NLP è più complessa (in figura 2.2 ne vediamo una semplificazione) e ha rappresentato la maggior parte del lavoro dell'attività di tesi. L'architettura del modello verrà trattata nel dettaglio con il proseguo della tesi.

2.2 Considerazioni e attività preliminari

Il modello proposto si basa su un'ampia gamma di considerazioni e di attività preliminari svolte sui dati, specialmente per quanto riguarda i documenti 10-K. I documenti 10-K, infatti, sono documenti che possono raggiungere dimensioni importanti, contengono molte informazioni superflue e, soprattutto, hanno dimensioni differenti l'uno dall'altro. Proprio per la lunghezza dei documenti 10-K si è incontrato il primo limite architetturale. Infatti, DistilBERT, che è il modello designato alla generazione dei *document embedding*, ha un limite massimo in input di 512 tokens ed è per questo si è dovuta studiare una strategia per aggirare questo limite. La strategia elaborata per aggirare questo limite architetturale di DistilBERT segue determinati step:

1. Selezione degli items che rappresentano in maniera migliore la stabilità di un'azienda. Gli items 1, 5, 7 si sono rivelati i migliori a questo proposito. Dopo questo step per ogni azienda si è quindi considerata la concatenazione di questi 3 items.

2. Sfruttando il fenomeno descritto in *Lazy prices* [2], quindi il fatto che la pigrizia porta i manager a modificare solo certe parti del 10-K dell'anno precedente, si sono estratte, mediante *document subtraction*, solamente le parti di testo aggiunte nell'anno preso in considerazione.

3. A questo punto i frammenti di testo ottenuti da questa analisi preliminare vengono suddivisi in chunks di 256 token ciascuno e vengono mantenuti solamente i chunks che esprimono un "sentimento" positivo o negativo. Questa *sentiment analisys* è stata effettuata mediante *FinBERT*.

Conclusa questa parte di pre-processing, i chunks vengono dati in input alla parte di NLP, nella quale DistilBERT genera per ogni chunk un embedding. Successivamente, mediante varie tecniche descritte nei prossimi capitoli, si vanno a ridurre gli n embedding dei chunks per ogni azienda in un unico vettore. A questo punto, il vettore ottenuto è dato in input ad una rete neurale che ha il compito di "comprimerlo", in modo tale che sia di più facile interpretazione per la rete di output. L'ultimo layer della rete di compressione è poi concatenato con la rete LSTM e, infine, la rete in output, considerando entrambe le parti, quella di analisi delle variabili finanziarie e la rappresentazione dei documenti, generata mediante la parte di NLP, darà un risultato binario.

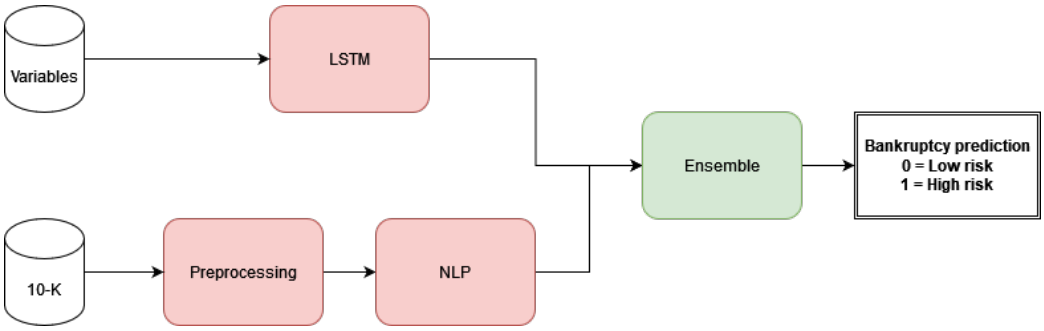

Figura 2.3: Immagine che rappresenta il contributo di entrambi i modelli (NLP e LSTM) per il task di classificazione binaria.

Capitolo 3

I dati

Il dataset iniziale [22] è composto da 6190 aziende appartenenti al mercato statunitense, quindi quotate in NASDAQ e NYSE, distribuite temporalmente dal 1999 al 2014. Per generare i vari set di dati (training-set, validation-set e test-set) si è deciso di suddividere temporalmente il dataset nel seguente modo:

- Il training-set fa riferimento ai dati dal 1999 al 2014.
- Il validation-set fa riferimento solamente al 2015.
- Il test-set fa riferimento ai dati dal 2016 al 2018.

Come si può immaginare, il dataset è fortemente sbilanciato, in quanto vi sono più aziende non fallite di quelle che sono andate in bancarotta. Di conseguenza, si è dovuto bilanciare il dataset sia in fase di training, in modo tale che entrambe le classi (fallita, sana) siano "imparate" allo stesso modo dal modello, sia in fase di test in modo tale da poter utilizzare come misura di performance l'accuratezza.

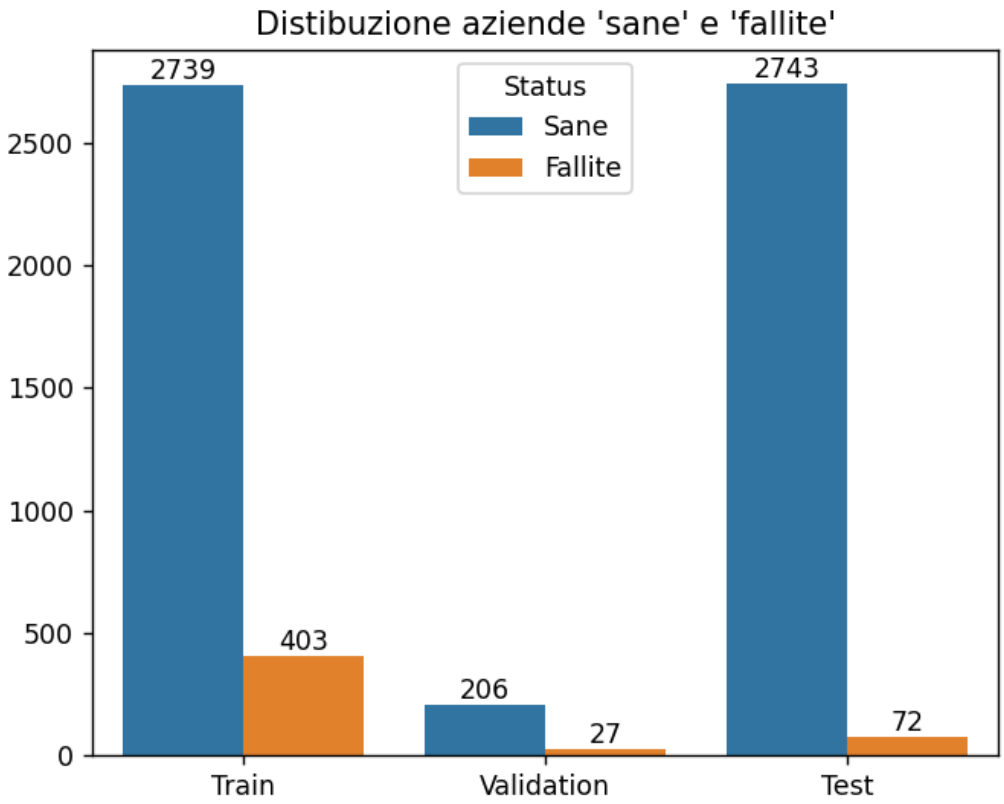

Figura 3.1: In figura è rappresentata la distribuzione di aziende "sane" e "fallite" nei vari set di dati.

Per il download dei dati, sia finanziari che dei report 10-k, si è utilizzata un'architettura distribuita utilizzando gli strumenti: Node.JS, MongoDB e Docker.

3.1 Estrazione variabili finanziarie

I dati finanziari, per quanto riguarda la parte elaborata dalla rete LSTM, sono stati estratti dal database WRDS *(**W**harton **R**esearch **D**ata **S**ervice)*, collaborando con la University of Florida. Più in particolare, si sono recuperati dal CRSP *(**C**harton for **R**esearch in **S**ecurity **P**rices)* i dati azionari mensili attraverso gli identificativi CIK *(**C**entral **I**ndex **K**ey)* estratti

dal WRDS. Questo è risultato possibile in quanto la SEC *(**S**ecurities and **E**xchange **C**ommission)* individua le aziende statunitensi mediante, appunto, il codice CIK. Successivamente, sono stati calcolati i ritorni annuali per l'anno fiscale a partire dal 1° di Aprile, in quanto i 10-K per quella data dovrebbero essere già stati pubblicati. Per ogni anno, dunque, sono disponibili i seguenti dati finanziari:

Current assets	Total current liabilities
Total assets	Net income
Cost of goods sold	Retained earnings
Total long term debt	Total receivables
Depreciation and ammortization	Total revenue
EBIT	Market values
EBITDA	Total liabilities
Gross profit	Net sales
Inventory	Total operating expenses

Tabella 3.1: Varibili finanziarie considerate.

3.2 Estrazione documenti 10-K

I dati, per quanto riguarda la parte di NLP, sono stati, appunto, tratti dai report 10-K. Un report 10-K è un report che ogni azienda (con reddito superiore a 10 milioni di dollari e una classe di titoli azionari detenuti da più di 2000 persone) è obbligata dalla *SEC* a pubblicare su base annuale e riassume le performance di una azienda sul mercato azionario, l'andamento del business e lo stato patrimoniale. I rapporti 10-K devono essere pubblicati entro 90 giorni dalla chiusura dell'anno fiscale. Questa scadenza si riduce a 75 giorni per aziende con un capitale flottante di oltre 700 milioni di dollari. La *SEC* rende disponibili i report 10-K, come tutti gli altri documenti, nella base di dati EDGAR accessibile mediante il sito web della *SEC*. Ogni 10-K è suddiviso in 15 parti, chiamate *items*, raggruppabili in 4 sezioni:

Item 1	Business	**PARTE 1**
Item 1A	Fattori di rischio	
Item 1B	Commenti del personale non risolti	
Item 2	Proprietà	
Item 3	Procedimenti legali	
Item 4	Informazioni sulle violazioni della sicurezza	
Item 5	Mercato	**PARTE 2**
Item 6	Dati finanziari consolidati	
Item 7	Discussione e analisi delle condizioni finanziarie e dei risultati delle operazioni	
Item 7A	Informazioni qualitative e quantitative sui rischi di mercato	
Item 8	Bilancio	
Item 9	Modifiche e disaccordi con i contabili sulla contabilità e informazioni finanziarie	
Item 9A	Controlli e procedure	
Item 9B	Altre informazioni	
Item 10	Amministratori, amministratori delegati e corporate governance	**PARTE 3**
Item 11	Compensi dirigenti	
Item 12	Titolarità di alcuni aventi diritto effettivo e questioni relative alla gestione e agli azionisti correlati	
Item 13	Determinate relazioni e transazioni correlate e indipendenza del direttore	
Item 14	Commissioni e servizi contabili principali	
Item 15	Allegati e firme degli schemi di bilancio	**PARTE 4**

Tabella 3.2: Struttura dettagliata di un documento 10-K.

Capitolo 4

Implementazione della rete LSTM

4.1 Architettura della rete

Per la parte di analisi di dati finanziari è stato scelto di utilizzare una rete LSTM Multi-head (**L**ong **S**hort-**T**erm **M**emory) (Hochreiter e Schmidhuber, 1997 [7]) che, come già detto in precedenza, è un modello che si adatta particolarmente bene all'analisi di dati che sono distribuiti su serie temporale, proprio come nel caso dei dati finanziari estratti. Il modello è così composto:

- Un **Input layer** di dimensione 18, pari al numero di features in ingresso.
- Un **LSTM layer** di dimensione 18, pari al numero di features in ingresso.
- Un **Concatenate layer** che permette di concatenare gli output delle varie teste LSTM in uno unico.
- Un **Dense layer** di dimensione 18, pari al numero di features in ingresso, in cui ogni neurone ha funzione di attivazione ReLU.
- Un **Dense output layer** di dimensione 2, con funzione di attivazione Softmax.

In figura 4.1 è riportato uno schema dettagliato dell'architettura proposta per il modello di analisi delle variabili finanziarie.

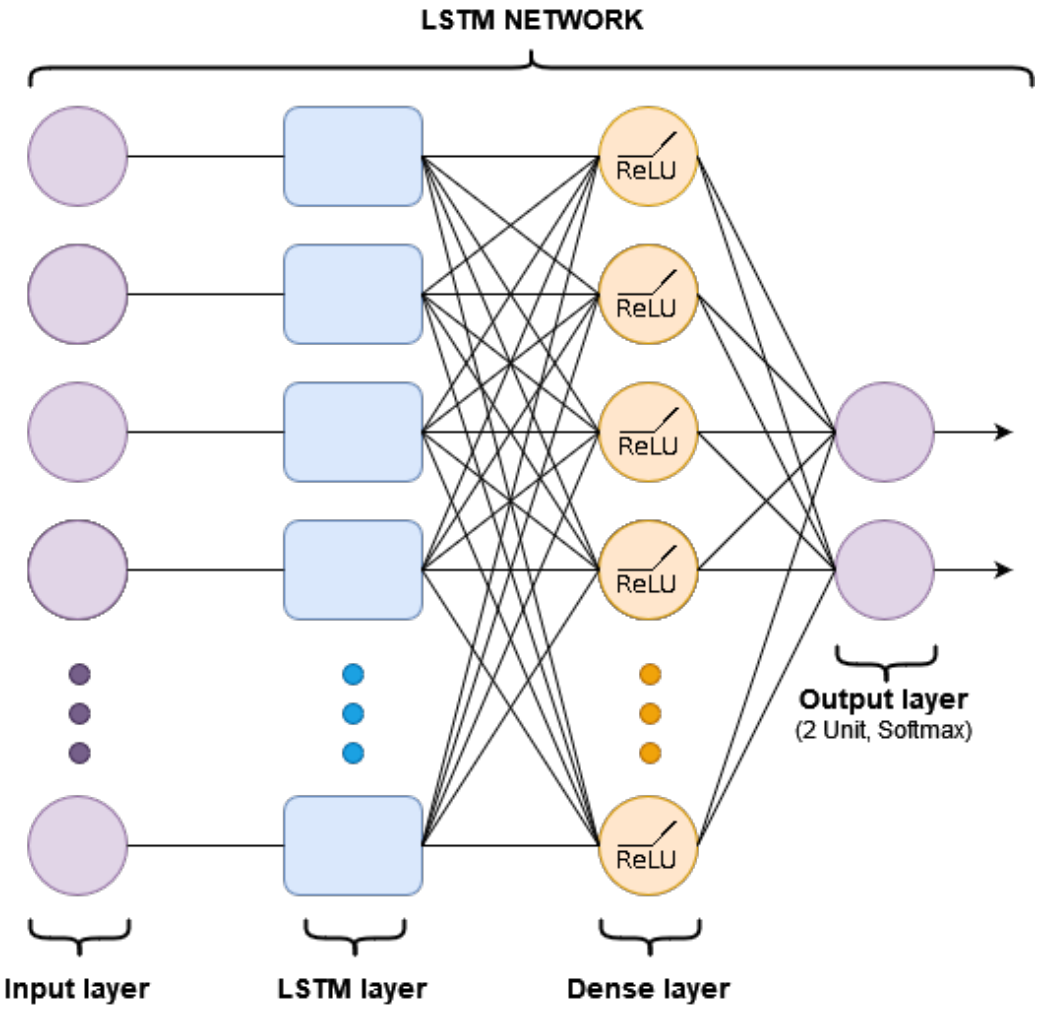

Figura 4.1: In questo schema è rappresentata in modo dettagliato l'architettura proposta per il modello di analisi delle variabili finanziarie.

4.2 Implementazione e fine-tuning del modello

Ogni testa della rete LSTM analizza la sequenza temporale per ciascuna feature. Questa rete, come tutti i modelli implemantati durante l'attività

di tesi, è stata sviluppata con la libreria `Keras` [23] presente nel modulo `Tensorflow` [24]. Per quanto riguarda i parametri, la rete LSTM utilizza come optimizer *Adam*[1], con learning rate pari a 0.0001 e come funzione di loss la *Binary cross entropy*[2]. Inoltre, è stato applicato come meccanismo di regolarizzazione l'*early stopping*[3], con un parametro di pazienza pari al 10% del numero di epoche di addestramento. Il *fine-tuning* della rete è stato effettuato mediante *grid-search* sui parametri: numero di celle per layer, numero di epoche e dimensione dei batch. Inizialmente si è stabilito che il numero di celle per layer che dava la miglior accuratezza era 14, dopodiché si è proseguita l'attività di fine-tuning mantenendo fisso questo parametro e si è ottenuto come valore ottimale 500 per il numero di epoche e 100 per la dimensione dei batch.

Hyper-parameter	**Best value**
Units	14
Epochs	500
Batch size	100
Early stopping patience	50

Tabella 4.1: Parametri calcolati mediante grid search.

Dato che il dataset è stato bilanciato, il modello è stato valutato sia sul validation-set, durante il fine-tunig, che successivamente sul test-set, utilizzando la misura di accuratezza.

[1]L'ottimizzazione Adam è basata sull'algoritmo di discesa del gradiente stocastico.

[2]La binary cross entropy è una funzione di loss utilizzata nei task di classificazione binaria.

[3]L'early stopping è una metodologia di regolarizzazione utilizzata per evitare l'overfitting, permette di interrompere l'addestramento prima che il modello vada in overfit.

Capitolo 5

Implementazione del modello di NLP

5.1 Pre-processing

Prima di iniziare a progettare il modello di NLP, a differenza della LSTM, si è dovuta svolgere una minuziosa operazione di pre-processing sui documenti 10-K, in modo tale da ridurre i dati mantenendo solo quelli di rilevanza per l'attività. Come già accennato nella sezione *"Considerazioni e attività preliminari"*, si sono svolte 3 attività principali per quanto riguarda il pre-processing: *scelta degli items migliori*, *document subtraction* e *sentiment analisys*.

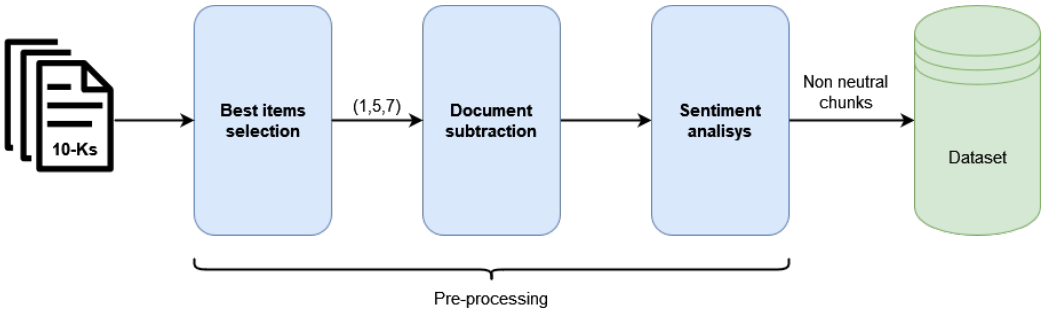

Figura 5.1: In figura è rappresentata l'attività di pre-processing svolta.

5.1.1 Scelta dei migliori items

Inizialmente si era pensato di utilizzare l'intero documento in input a DistilBERT. Ciò non è stato possibile, in quanto DistilBERT ha un limite di token in input di 512 e, come vediamo in figura 5.2, i vari items hanno un elevato numero di parole. Perciò si è iniziato selezionando gli items del documento 10-K che generalmente risultano più significativi nell'identificare la salubrità di un'azienda.

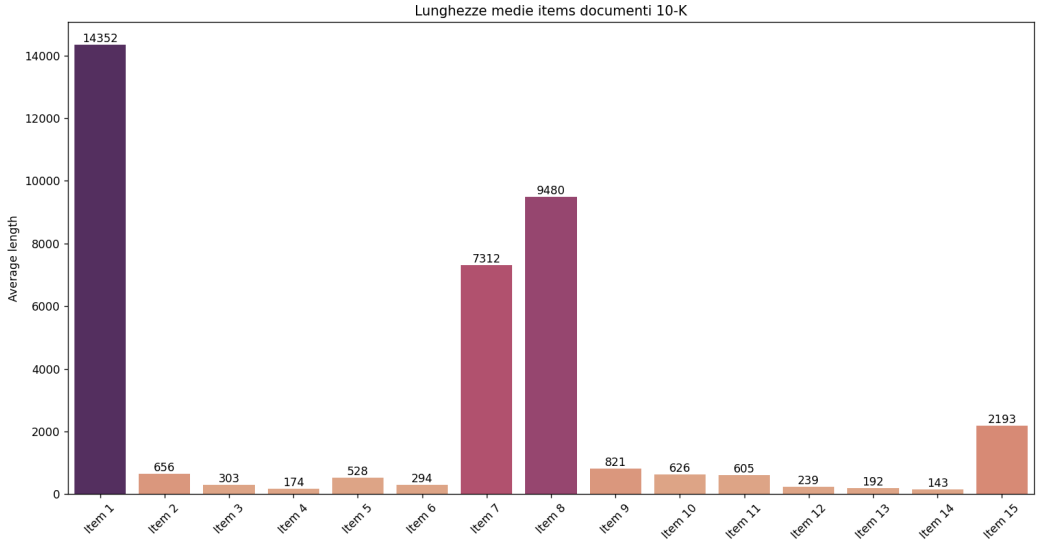

Figura 5.2: In figura è rappresentato il numero di parole in media per ogni item nei report 10-K.

Questa analisi è stata effettuata utilizzando due tecniche, che possono sembrare semplici ma che sono risultate efficaci:

- Usando la codifica *Bag-of-Words*[1] e l'utilizzo di modelli classificatori quali *Logistic regression* e *Gradient boosting*.
- Utilizzo del dizionario proposto da Loughran e McDonald [25], valutato successivamente mediante *Logistic regression*.

[1]Il modello Bag-of-Words è un metodo utilizzato nell'Information Retrieval e nel NLP per rappresentare documenti ignorando l'ordine delle parole.

In figura 5.3 è riportata la sequenza di attività svolte e appena descritte. Queste operazioni sono operazioni preliminari che non fanno parte dell'addestramento del modello finale, ma sono essenziali al suo funzionamento.

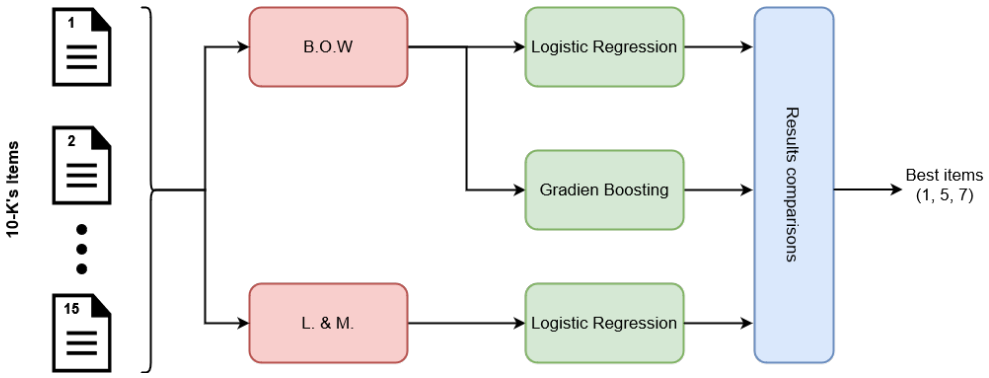

Figura 5.3: In figura è rappresentata la pipeline delle attività svolte per la ricerca degli items più significativi.

Analisi con il modello Bag-of-Words

La codifca con il modello Bag-of-Words richiede che sia effettuata un'operazione di *stemming*, che consiste nel ridurre ogni parola alla sua radice in modo che parole con diverse declinazioni siano considerate come stessa parola. Per esempio, le parole computer, computing, compute, computed e computable dopo l'operazione di stemming rimangono tutte *comput-*, che è la radice di queste parole. Successivamente sono state estratte dal train-set le 10000 parole più frequenti e sono state utilizzate come vocabolario per generare le *features* dei tre set (train, validation, test) in input ai modelli (Logistic regression e Gradient boosting) per l'addestramento. Il train e il validation set sono stati bilanciati e i risultati calcolati su una media di 10 iterazioni in modo da compensare la pseudo-randomicità dell'inizializzazione dei pesi durante l'addestramento. I risultati ottenuti sono che gli item 1, 5 e 7 sono gli item più significativi per quanto riguarda la salubrità dell'azienda.

Analisi con il dizionario di Loughran & McDonald

In alternativa all'analisi basata sul modello Bag-of-Words si è effettuata un'analisi mediante il vocabolario di Loughran e McDonald [25], il quale è specializzato in argomenti finanziari. Per ciascun item sono state prodotte alcune statistiche:

- Numero di parole.
- Percentuale di parole che esprimono un sentimento negativo.
- Percentuale di parole che esprimono un sentimento positivo.
- Percentuale di parole che esprimono un sentimento di incertezza.
- Percentuale di parole che esprimono un sentimento litigioso.
- Percentuale di parole che esprimono un ordine (verbi modali forti).
- Percentuale di parole che esprimono un suggerimento (verbi modali deboli).
- Percentuale di parole che esprimono un sentimento di imposizione.
- Numero di lettere alfabetiche.
- Numero di cifre.
- Numero di numeri.
- Media del numero di sillabe per parola.
- Media della lunghezza delle parole.

Queste statistiche sono utilizzate come *features* in input ad un modello di *regressione logistica* che, come esito, ha rivelato, anche in questo caso, che gli items più rilevanti sono l' 1, il 5 e il 7.

Considerazioni sui risultati ottenuti

Visti i risultati ottenuti, si può effettivamente notare che gli items 1, 5, 7 sono di rilevanza in base agli argomenti che trattano. Infatti, nell'item 1 viene descritta l'azienda e il mercato in cui si colloca, l'item 5 offre informazioni relative al mercato, sugli azionisti e gli acquisti di titoli azionari e, infine, l'item 7 mostra i rischi del mercato in cui si colloca l'azienda, illustrandone le prestazioni e le prospettive future.

5.1.2 Document subtraction

In precedenza è già stato accennato che la pigrizia dei manager li porta a riutilizzare i documenti 10-K dell'anno precedente (*Lazy prices, Cohen, 2020* [2]), per questo si effettua un'operazione di document subtraction con il fine di tenere solamente le parti modificate nell'ultimo documento da quello dell'anno precedente. Per svolgere questa operazione si è utilizzata la libreria `difflib` di python che, mediante la classe `SequenceMatcher`, permette di individuare, date due sequenze di lunghezza arbitraria, le parti uguali, le parti modificate, le parti aggiunte e quelle cancellate. Lo script che si occupa di effettuare questa operazione è stato sviluppato per funzionare in 2 modalità: una prevede la rimozione delle *Stopwords*, mentre l'altra le mantiene. È stato fatto ciò per valutare le performance del modello di NLP in entrambe le casistiche.

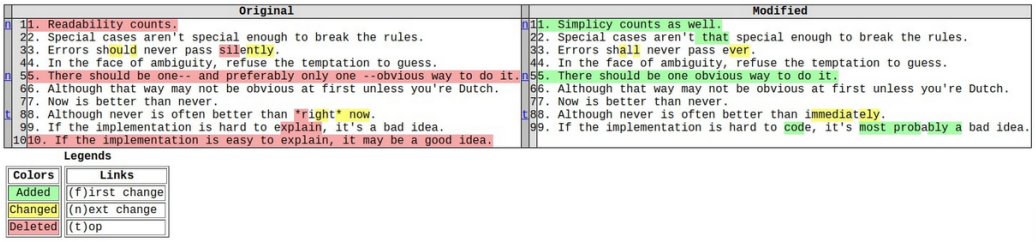

Figura 5.4: Rappresentazione dell'utilizzo della classe `SequenceMatcher` del modulo `difflib`.

5.1.3 Sentiment analisys

La fase finale di pre-processing è la fase di *sentiment analisys*. Si esegue questa fase in quanto, dopo la document subtraction, abbiamo ancora comunque testi di lunghezza arbitraria. In questa fase si divide ogni testo in *chunks* (frammenti) di 256 parole l'uno. La scelta di utilizzare 256 token per chunk e non 512 (che è il limite di input di DistilBERT usato nelle fasi successive) deriva solamente da un limite computazionale, infatti, più token si hanno in input più la complessità computazionale aumenta. Una volta suddiviso il testo in vari chunks si utilizza FinBERT (BERT addestrato in modo specifico su documenti finanziari, approfondito nella sezione *FinBERT*) per selezionare solamente i chunks che esprimono un sentimento positivo o negativo e, quindi, scartare quelli neutri. A questo punto viene generato un dataset con tutti i chunks, che possono essere molteplici per azienda.

CIK	**Status**	**Chunk**
3197	0	various end markets provides us with multiple sources...
3197	0	the performance of these functions and could...
3197	0	reputation the everevolving threats mean...
2034	1	revenue and earnings due to the persistent...

Tabella 5.1: Breve estratto di alcune righe del dataset dopo il pre-processing.

5.1.4 Considerazioni sul pre-processing

Prima di passare alla parte principale, per quanto riguarda il modello di NLP, è bene fare alcune considerazioni sulla riduzione del dataset durante il pre-processing: per forza di cose alcune aziende vanno perse. Notiamo che, mantenendo le stopwords, riusciamo ad avere un dataset più ampio. Infatti FinBERT, in presenza di stopwords, performa meglio (le stopwords aiutano a comprendere il contesto nel caso dei *Transofrmers*). Di conseguenza, riesce a identificare una polarità per più chunks e, quindi, rimarranno più aziende e più chunks per ogni azienda. Le tabelle riportate di seguito riportano

un'analisi quantitativa per quanto riguarda la perdita di dati durante il pre-processing.

Set	Item	Fase pre-processing	Stopwords	Aziende fallite
TRAIN SET	ITEM 1	Iniziale	-	403
		Document subtraction	SI	361
			NO	346
		Sentiment analisys	SI	321
			NO	203
	ITEM 5	Iniziale	-	401
		Document subtraction	SI	350
			NO	301
		Sentiment analisys	SI	44
			NO	17
	ITEM 7	Iniziale	-	403
		Document subtraction	SI	352
			NO	328
		Sentiment analisys	SI	326
			NO	237

Tabella 5.2: Come decrescono le aziende fallite ad ogni fase del pre-processing sul train set.

Set	Item	Fase pre-processing	Stopwords	Aziende fallite
VALIDATION SET	ITEM 1	Iniziale	-	27
		Document subtraction	SI	25
			NO	25
		Sentiment analisys	SI	24
			NO	19
	ITEM 5	Iniziale	-	27
		Document subtraction	SI	23
			NO	24
		Sentiment analisys	SI	5
			NO	0
	ITEM 7	Iniziale	-	27
		Document subtraction	SI	21
			NO	22
		Sentiment analisys	SI	20
			NO	18

Tabella 5.3: Come decrescono le aziende fallite ad ogni fase del pre-processing sul validation set.

Set	Item	Fase pre-processing	Stopwords	Aziende fallite
TEST SET	ITEM 1	Iniziale	-	72
		Document subtraction	SI	60
			NO	59
		Sentiment analisys	SI	57
			NO	41
	ITEM 5	Iniziale	-	70
		Document subtraction	SI	57
			NO	45
		Sentiment analisys	SI	13
			NO	2
	ITEM 7	Iniziale	-	71
		Document subtraction	SI	53
			NO	56
		Sentiment analisys	SI	49
			NO	35

Tabella 5.4: Come decrescono le aziende fallite ad ogni fase del pre-processing sul test set.

5.2 Implementazione di DistilBERT

Una volta svolto il pre-processing, ci troviamo con un dataset diviso in train, validation e test, composto da vari chunks, di lunghezza non superiore a 256 token, per ogni azienda. DistilBERT è stato utilizzato in quanto è computazionalmente più leggero di BERT e sacrifica di poco le performance. Nonostante ciò, rimane comunque un modello pesante con circa 66 milioni di parametri. Nel modello di NLP proposto DistilBERT è utilizzato per l'estrazione dei *chunks embedding*, ovvero dell'estrazione, per ogni chunk, di un vettore di embedding.

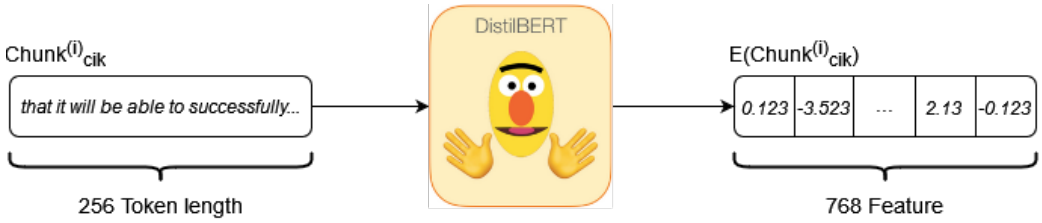

Figura 5.5: In figura è riportato uno schema semplificato di come avviene la trasformazione da dati testuali a features numeriche mediante DistilBERT.

5.2.1 Estrazione dei vettori di embedding

Il processo di estrazione degli embedding inizia con la fase di *tokenization*, nella quale si aggiunge, per ogni chunk, il token speciale [CLS] e il token di separazione [SEP]. Fatto ciò, ogni token viene convertito in un identificativo, dopodiché il chunk tokenizzato viene dato in input al Trasformer.

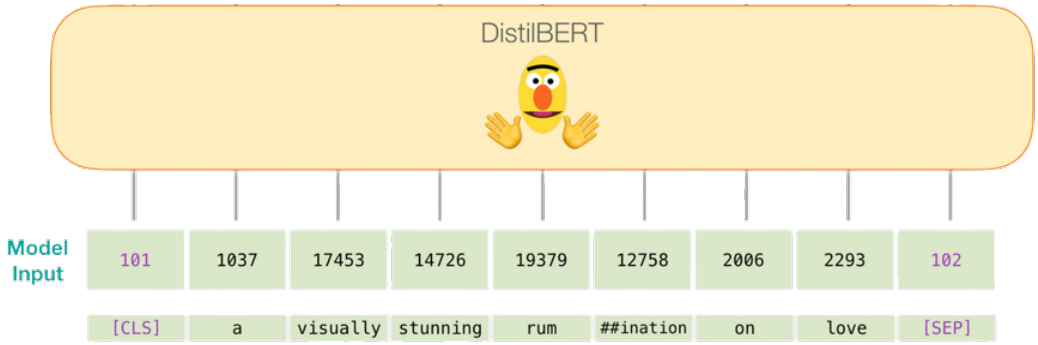

Figura 5.6: In figura è riportato il meccanismo di tokenizzazione sugli input di DistilBERT.

Una volta che DistilBERT processa gli input, per ottenere gli embedding occorre accedere all'ultimo *hidden state* di DistilBERT, dove sono memorizzati, per ogni token, i vettori di embedding lunghi 768. Il `Last_hidden_state` di DistilBERT è un *Tensore tridimensionale*, dove sull'asse x abbiamo il numero di token in un chunk, sull'asse y abbiamo il numero di chunks e

sull'asse z abbiamo la dimensione dell'embedding. Per i nostri scopi è importante solo l'embedding del token [CLS] perchè, come già detto in precedenza, corrisponde all'embedding dell'intera frase. Per estrarre l'embedding del token [CLS] dobbiamo accedere ad ogni riga in posizione 0, quindi accediamo a `Last_hidden_states[i][0]` e così otteniamo gli embedding dei token [CLS] per ogni chunk.

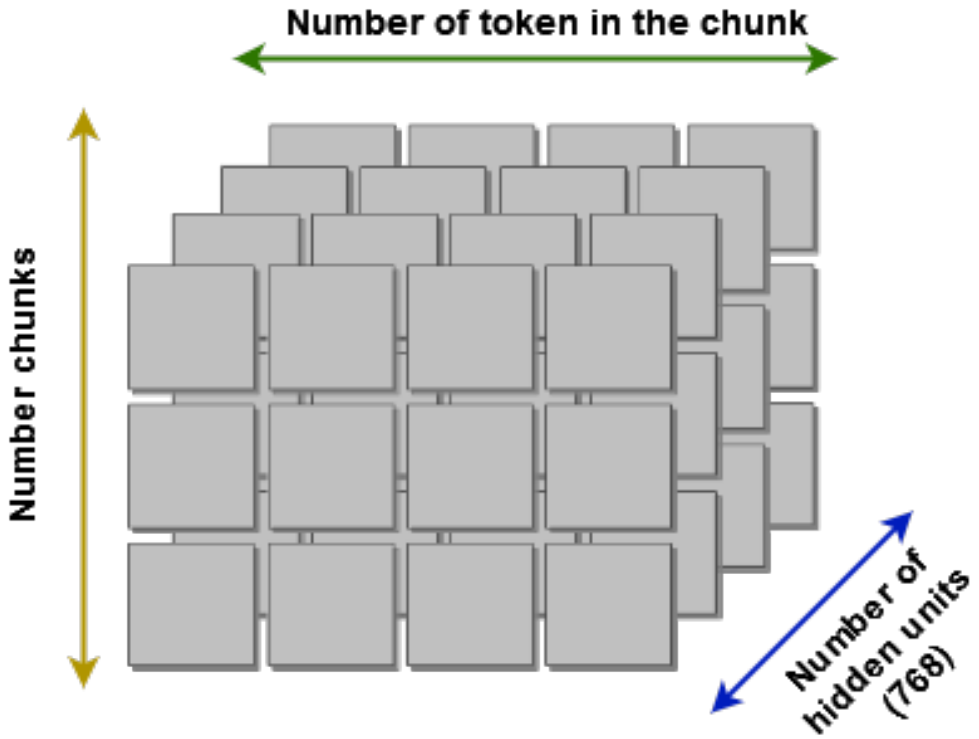

Figura 5.7: In figura è riportata la struttura del tensore di output di DistilBERT.

Fatto ciò, quello che abbiamo sono, per ogni azienda, tanti vettori di embedding quanti erano i chunks. A questo punto sorge, però, un problema: come è possibile fare una predizione su un'azienda se il numero di *chunks embedding* è variabile per ogni azienda? A primo impatto si potrebbe pensare che fare una predizione su ogni chunk e successivamente fare *hard voting*[2]

[2]Per hard voting si intende una predizione a voto di maggioranza tra più predizioni.

potrebbe essere una soluzione valida, in quanto, per esempio, potremmo usare come features i chunks positivi, quelli negativi e i chunks totali. Il fatto è che in questo modo il modello finale non risulta più addestrabile end-to-end e, inoltre, le features estratte sono povere di significato per un modello di intelligenza artificiale, perciò una parte consistente dell'attività di tesi è stata la ricerca di tecniche più sofisticate per la risoluzione di questo problema.

5.3 Unione degli embedding

L'idea è quella di unire in un unico vettore, che mantenga la stessa dimensione dei singoli *chunks embedding*, gli *n chunks embedding* per ogni azienda, in modo tale che il vettore ottenuto sia l'embedding della singola azienda. In questo modo si avranno tanti embedding quante sono le aziende.

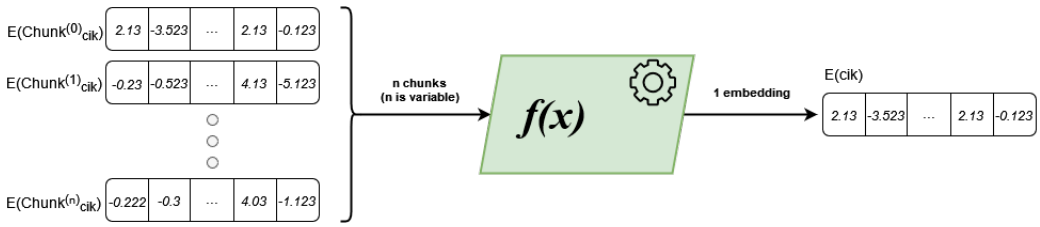

Figura 5.8: In figura è riportato lo schema del risultato che si vuole ottenere.

Le funzioni considerate a questo proposito sono già state presentate nel capitolo *"Stato dell'arte"* e sono: il *prodotto di Hadamard* e la *media tra vettori*.

5.3.1 Riduzione con prodotto di Hadamard

Come specificato in precedenza, per ridurre più vettori in uno solo mediante il prodotto di Hadamard occorre applicarlo iterativamente. Quindi, facendo riferimento ai *chunks embedding*, abbiamo che il prodotto di Hadamard tra

due chunks i e j è definito come:

$$\begin{bmatrix} E_0(chunk_{cik}^{(i)}) \\ E_1(chunk_{cik}^{(i)}) \\ \dots \\ E_{767}(chunk_{cik}^{(i)}) \end{bmatrix} \boxdot \begin{bmatrix} E_0(chunk_{cik}^{(j)}) \\ E_1(chunk_{cik}^{(j)}) \\ \dots \\ E_{767}(chunk_{cik}^{(j)}) \end{bmatrix} = \begin{bmatrix} E_0(chunk_{cik}^{(i)}) \cdot E_0(chunk_{cik}^{(j)}) \\ E_1(chunk_{cik}^{(i)}) \cdot E_1(chunk_{cik}^{(j)}) \\ \dots \\ E_{767}(chunk_{cik}^{(i)}) \cdot E_{767}(chunk_{cik}^{(j)}) \end{bmatrix} \tag{5.1}$$

Per ottenere E_{cik}, ovvero l'embedding per un'azienda, si prende quindi ogni embedding e si esegue il prodotto Hadamard fino a che non ci si riduce ad un unico vettore (E_{cik}). Più formalmente, si avrà:

$$E_{\text{cik}} = \prod_{i=0}^{n} E(chunk_{\text{cik}}^{(i)}) = E(chunk_{\text{cik}}^{(0)}) \boxdot E(chunk_{\text{cik}}^{(1)}) \boxdot \dots \boxdot E(chunk_{\text{cik}}^{(n)}) \tag{5.2}$$

dove n è il numero di chunk, variabile per ogni azienda.

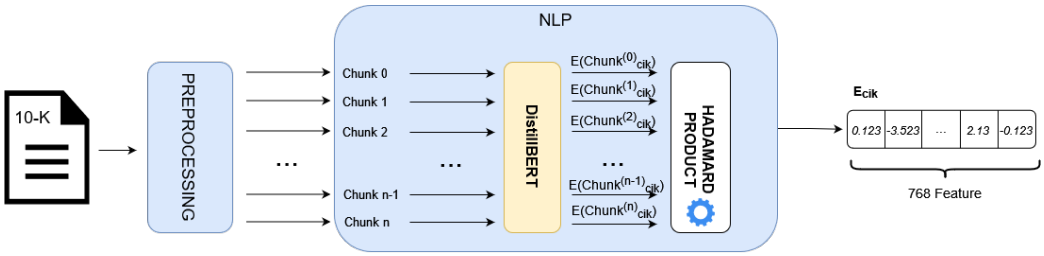

Figura 5.9: In figura è riportato uno schema dell'architettura descritta fino ad ora, considerando il prodotto di Hadamard.

5.3.2 Riduzione con media tra vettori

Come operazione alternativa al prodotto di Hadamard abbiamo la media tra vettori che, facendo riferimento ai *chunk embeddings*, possiamo formalizzare nel seguente modo:

$$E_{cik} = \frac{1}{n} \sum_{i=0}^{n} E(chunk_{cik}^{(i)}) \tag{5.3}$$

Nel nostro caso, la somma tra due *chunk embedding* generici i e j è definita come:

$$\begin{bmatrix} E_0(chunk_{cik}^{(i)}) \\ E_1(chunk_{cik}^{(i)}) \\ \dots \\ E_{767}(chunk_{cik}^{(i)}) \end{bmatrix} + \begin{bmatrix} E_0(chunk_{cik}^{(j)}) \\ E_1(chunk_{cik}^{(j)}) \\ \dots \\ E_{767}(chunk_{cik}^{(j)}) \end{bmatrix} = \begin{bmatrix} E_0(chunk_{cik}^{(i)}) + E_0(chunk_{cik}^{(j)}) \\ E_1(chunk_{cik}^{(i)}) + E_1(chunk_{cik}^{(j)}) \\ \dots \\ E_{767}(chunk_{cik}^{(i)}) + E_{767}(chunk_{cik}^{(j)}) \end{bmatrix} \tag{5.4}$$

Anche in questo caso lo scopo di ottenere un vettore di 768 per ogni azienda è stato raggiunto.

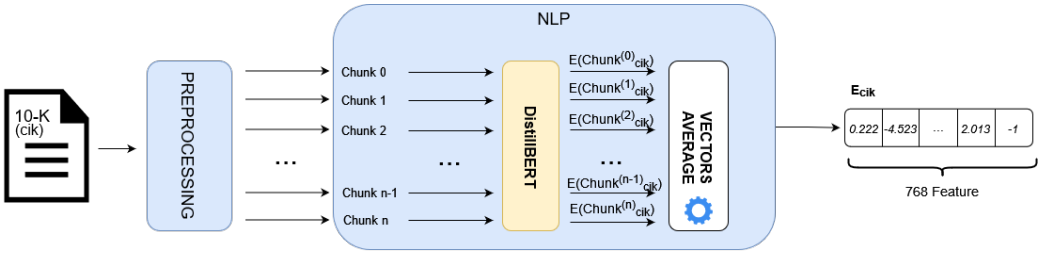

Figura 5.10: In figura è riportato uno schema dell'architettura descritta fino ad ora, considerando la media tra vettori.

5.4 Vantaggi e limiti delle tecniche provate

Iniziamo col dire che entrambe queste tecniche sono preferibili rispetto all'approccio accennato in precedenza con il quale, mediante hard-voting, si vanno ad estrarre alcune features in quanto, mediante le due tecniche proposte, il modello è addestrabile end-to-end. Inoltre, è da considerare il fatto che un modello di intelligenza artificiale si addestra in maniera migliore con vettori numerici che con features "human-friendly", come possono essere quelle estratte mediante hard-voting. Sia l'unione dei vettori basata sul prodotto di Hadamard che sulla media hanno alcuni pregi e difetti.Il prodotto di Hadamard rende di gran lunga più stabile l'addestramento del modello, infatti, il modello né cade in overfitting né in underfitting mentre, utilizzando la

media, il modello tende ad avere problemi di overfitting. Il problema, per quanto riguarda il prodotto di Hadamard, è venuto alla luce quando, utilizzando il dataset generato senza rimuovere le stopwords, quindi nettamente più ampio (più chunks per aziende), si è notato che i vettori risultanti dal prodotto contenevano features con valori numericamente importanti. Questo obbliga ad utilizzare tecniche di *feature scaling*[3]. Inoltre, per ovviare a questo problema per le aziende che avevano più di un certo numero di chunks, si sono scelti casualmente solo alcuni di questi in modo da ridurne il numero. È abbastanza banale comprendere il perché del fatto per cui si presenta questo problema con il prodotto di Hadamard, basti pensare che moltiplicando per molte volte numeri compresi tra 0 e 1 otteniamo numeri molto piccoli, mentre, moltiplicando numeri maggiori di 1 o minori di 0 otteniamo numeri molto grandi in positivo o negativo. Avere valori grandi nel dataset può portare al problema del *exploding gradient*, problema per il quale l'aggiornamento dei pesi avviene per quantità elevate e in maniera incontrollata. Una tecnica per compensare questo problema è, per esempio, la *l2 regularization*.

5.5 Implementazione di Doc2Vec

Come modello di confronto, alternativo a quello basato su DistilBERT, si è scelto di utilizzare Doc2Vec della libreria `Gensim`. Il modello è stato addestrato sul train-set effettuando un join di tutti i chunks per ogni azienda in modo da ricostruire il 10-K. Per effettuare un confronto alla pari con il modello basato su DistilBERT + Hadamard e con il modello DistilBERT + media tra vettori, Doc2Vec è stato configurato in modo da dare in output un vettore di 768 per ogni azienda. Doc2Vec è un modello che, per lavorare, necessita della rimozione delle stopwords, di conseguenza si sono usati i dataset senza stopwords.

[3]Nel machine learning il feature scaling permette di normalizzare il range di variazione delle feature di un dataset.

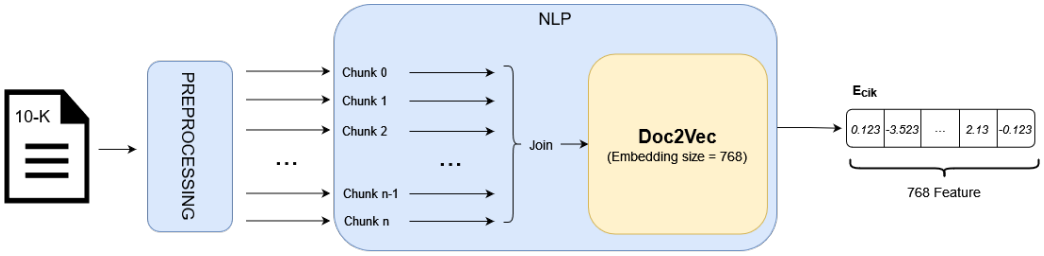

Figura 5.11: In figura sono è riportato uno schema dell'architettura descritta fino ad ora utilizzando però Doc2Vec.

Parametro	**Valore**
Vector_size	768
Window	2
Min_count	1
Workers	4

Tabella 5.5: Parametri Doc2Vec

5.6 Rete di compressione

Come riportato nello schema semplificato dell'architettura totale del modello, tra il modello di NLP e il layer di concatenazione con la rete LSTM vi è una rete con la funzione di "comprimere" le features in uscita dal modello di NLP. Una consistente parte dell'attività di tesi è stata dedicata alla progettazione e al fine-tuning di questa rete. In uscita dal modello di NLP abbiamo 768 features, che sono un numero elevato da gestire per una rete neurale se quest'ultima deve anche gestire la parte di LSTM. Per questo, la rete neurale che va ad elaborare gli embedding uscenti dalla parte di NLP, ha una struttura ad "imbuto", quindi che ha un numero decrescente di neuroni per ogni layer più ci si avvicina all'output layer, in modo tale che questa rete "impari" una rappresentazione degli embedding composta da meno features.

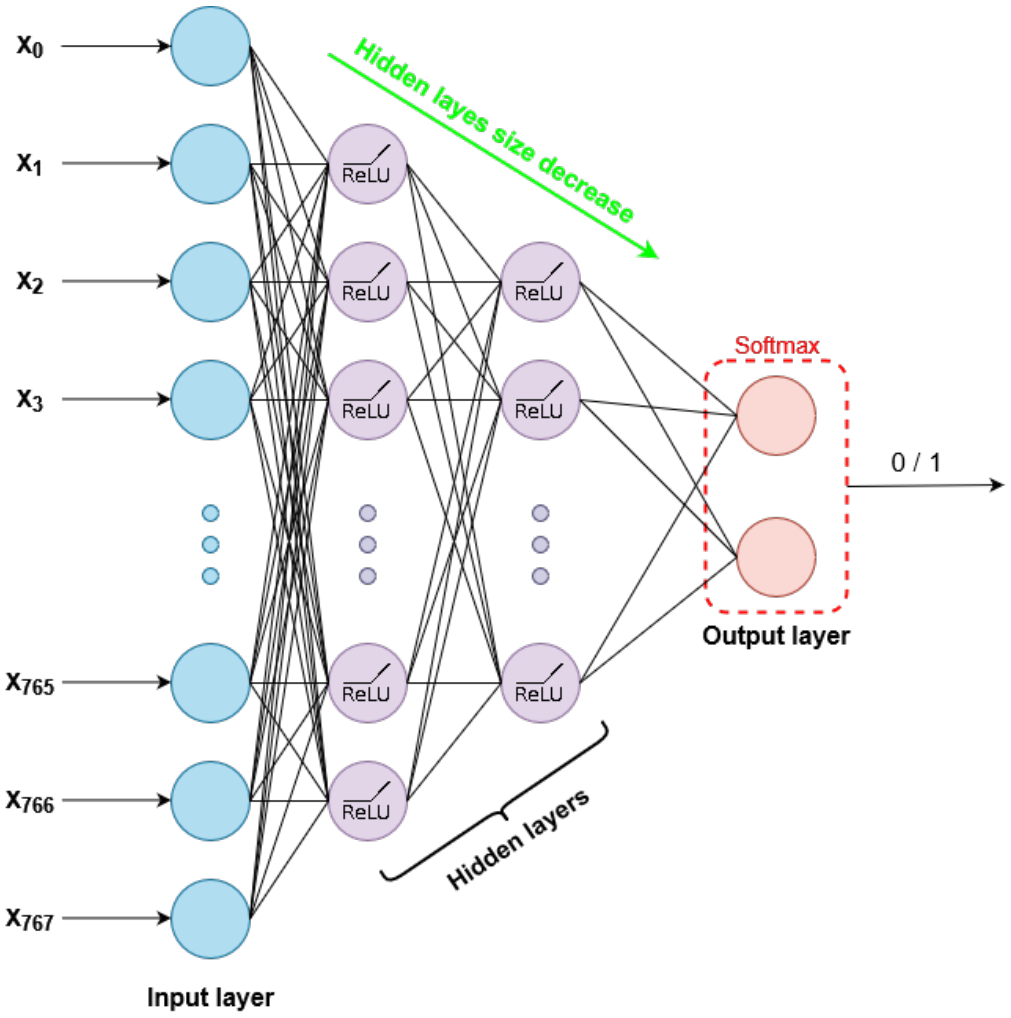

Figura 5.12: In figura è riportato uno schema dettagliato della rete di compressione.

5.6.1 Progettazione della rete

La rete di compressione è stata progettata come se dovesse effettuare la predizione finale (azienda fallita, azienda sana). Poi, una volta trovata la configurazione migliore (con accuratezza più alta) mediante il fine-tuning, si è rimosso l'output layer e si è andato a concatenare nel layer concatenate l'ultimo hidden layer. L'operazione di progettazione è stata effettuata 5 volte, infatti, alla fine si sono progettate 5 reti, una per ogni configurazione provata

per la parte di NLP:

- DistilBERT + prodotto di Hadamard, mantenendo le stopwords.
- DistilBERT + prodotto di Hadamard, rimuovendo le stopwords.
- DistilBERT + media tra vettori, mantenendo le stopwords.
- DistilBERT + media tra vettori, rimuovendo le stopwords.
- Doc2Vec, rimuovendo le stopwords.

Ognuna di queste reti ha un'architettura e dei parametri differenti. Questa decisione di progettare queste reti è stata fatta sia per confrontarle solo sulla parte di NLP, sia provando ciascuna di queste reti nel modello finale, per trovare la configurazione più performante. L'operazione di fine-tuning è stata effettuata mediante 3 metodologie principali:

- ***N****etwork* ***S****earch* ***A****rchitecture*: si tratta del primo algoritmo eseguito. Mediante un meccanismo a tentativi prova, senza particolari parametri, varie configurazioni per quanto riguarda gli hidden layers della rete. Più in particolare, trova sia il numero migliore di hidden layer che il numero di neuroni per ogni hidden layer.
- *Random search*: mediante una random search si provano varie configurazioni di parametri sulla rete che ha dato il miglior esito sulla N.A.S.
- *Grid search*: infine, si esegue una grid search in intervalli di parametri vicini a quelli trovati dalla random search, sempre utilizzando la rete che ha dato il miglior esito sulla N.A.S.

Alla fine di questi 3 step abbiamo la configurazione della rete migliore. Questa operazione è stata svolta 5 volte per tutte le configurazioni citate prima. Nella tabella sottostante sono riportati i parametri della rete migliori per ogni configurazione.

Configurazione modello	**Stopwords**	**Miglior configurazione**
DistilBERT + Hadamard	SI	1 Hidden layers: (30) Es. Patience: 30 Batch size: 20 Epochs: 568 Learning rate: 0.001
	NO	3 Hidden layers: (741, 329, 73) Es. Patience: 74 Batch size: 66 Epochs: 705 Learning rate: 0.00001
DistilBERT + Media vettori	SI	1 Hidden layers: (5) Es. Patience: 10 Batch size: 30 Epochs: 326 Learning rate: 0.0001
	NO	1 Hidden layers: (5) Es. Patience: 10 Batch size: 30 Epochs: 326 Learning rate: 0.0001
Doc2Vec	NO	3 Hidden layers: (739, 341, 23) Es. Patience: 30 Batch size: 40 Epochs: 400 Learning rate: 0.005

Tabella 5.6: Parametri migliori per ogni rete per ciascuna delle configurazioni proposte.

Capitolo 6

Modello finale

Progettate la parte di analisi di dati finanziari e la parte di analisi dei report 10-K, occorre unire questi due modelli. Questo viene fatto mediante una rete neurale che comprende un layer *CONCATENATE* dove si vanno a combinare i due modelli, oltre che ad un layer *DENSE* da 20 neuroni con attivazione ReLU e un output layer da 2 unità con attivazione Softmax.

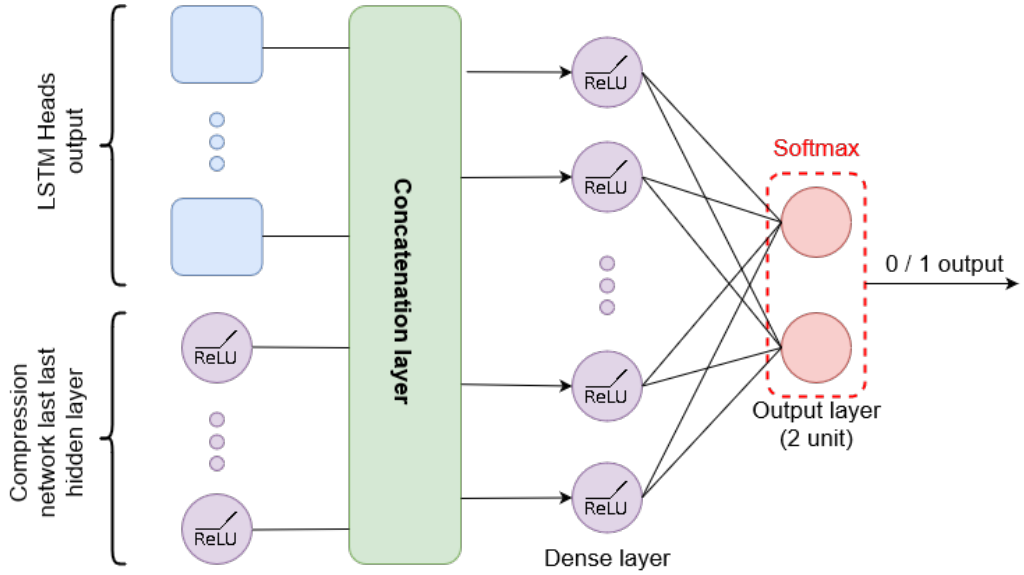

Figura 6.1: In questo schema è rappresentata in dettaglio la rete in cui si concatenano i modelli LSTM e di NLP.

6.1 Fine-tuning

Anche per il modello finale è stata svolta un'operazione di fine-tuning per la rete di output e, anche questa volta, come per la rete di compressione, si sono progettati 5 modelli in base alle configurazioni della parte di NLP a fine di confronto. Il fine-tuning è stato svolto mediante *Random search*. Di seguito vengono riportati i parametri ottimali per ogni configurazione.

Configurazione modello	**Stopwords**	**Miglior configurazione**
DistilBERT + Hadamard	SI	Es. Patience: 99 Batch size: 24 Epochs: 548 Learning rate: 0.001
	NO	Es. Patience: 81 Batch size: 4 Epochs: 691 Learning rate: 0.00001
DistilBERT + Media vettori	SI	Es. Patience: 82 Batch size: 14 Epochs: 745 Learning rate: 0.001
	NO	Es. Patience: 82 Batch size: 14 Epochs: 745 Learning rate: 0.001
Doc2Vec	NO	Es. Patience: 77 Batch size: 12 Epochs: 599 Learning rate: 0.0001

Tabella 6.1: Parametri migliori per ogni rete per ciascuna delle configurazioni proposte.

6.2 Architettura finale

Per l'addestramento finale del modello, ovviamente, occorre sincronizzare i dati finanziari con i dati dei 10-K. Questo viene fatto facendo corrispondere i CIK delle aziende dei due dataset. In figura 6.2 viene riportata l'architettura finale del modello con un certo livello di dettaglio.

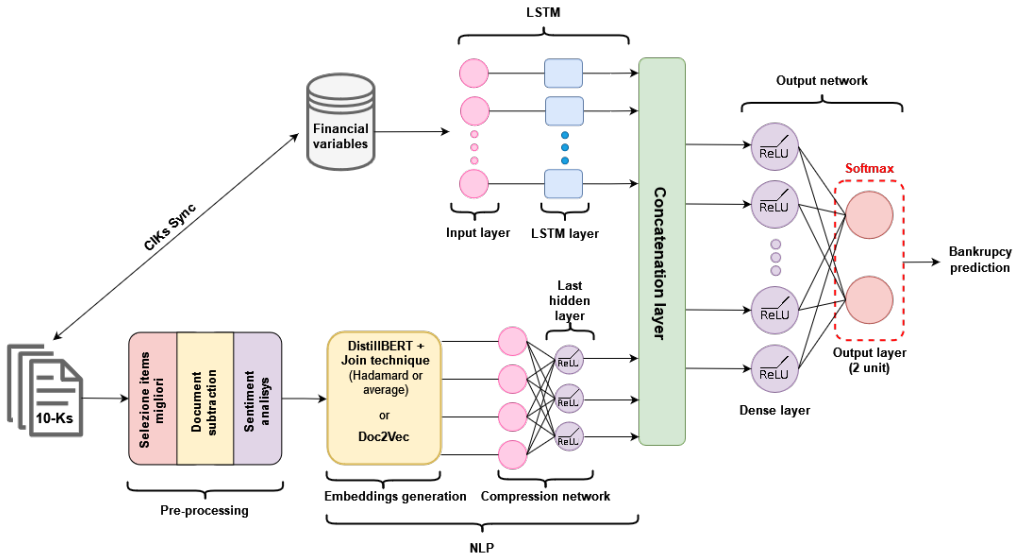

Figura 6.2: In figura è riportato lo schema dettagliato del modello finale progettato.

Capitolo 7

Risultati

In questo capitolo si illustrano i risultati ottenuti. In particolare, si effettua un confronto tra le performance del modello finale, in tutte le sue configurazioni, con i singoli moduli che lo compongono, ovvero la rete LSTM e la parte di NLP. Inoltre, si riportano i risultati ottenuti mediante lo Z-Score di Altman.

7.1 Metriche utilizzate

Per valutare ogni modello utilizzato è stata utilizzata esclusivamente la metrica di accuratezza, in quanto i dataset sono sempre stati bilanciati. Questa scelta deriva dal fatto che la metrica di accuratezza è una metrica che esprime con estrema semplicità le performance dei modelli. L'accuratezza si calcola come

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{7.1}$$

dove TN sono i *True Negatives*, TP sono i *True Positive*, FP sono i *False Positive* e, infine, FN i *False Negatives*.

7.2 Z-Score di Altman

Storicamente, per predire la bancarotta, si fa riferimento al punteggio (Z-Score) proposto nel 1968 da Altman in *"Financial ratios, discriminant analysis and the prediction of corporate bankruptcy"* [1]. I risultati ottenuti si sono voluti confrontare anche con questo punteggio per avere un paragone con un modello ormai consolidato. Lo Z-Score di Altman viene calcolato a partire dai dati di bilancio secondo la formula:

$$S = 1.2a + 1.4b + 3.3c + 0.999\frac{d}{e} + 0.6\frac{f}{g} \tag{7.2}$$

dove le variabili che appaiono hanno i seguenti significati:

- a = (attivo corrente - passivo corrente) / attivo totale
- b = utili non distribuiti / attivo totale
- c = utile ante interessi e imposte / attivo
- f/g = valore di borsa del capitale netto / valore di libro del debito
- d/e = vendite / attivo totale

In base al punteggio S le aziende si classificano nel seguente modo:

$S \leq 1.8$	Alta probabilità di fallimento
$1.8 < S < 3$	Non definito, ulteriori analisi necessarie
$S \geq 3$	Bassa probabilità di fallimento

Tabella 7.1: Classificazione aziende in base al punteggio Z-Score di Altman.

Nei punteggi riportati nelle prossime sezioni, si sono considerate le aziende con punteggi incerti, tra 1.8 e 3, come aziende con bassa probabilità di bancarotta.

7.3 Performance ottenute

Le performance sono state su una media di 20 test-set, ognuno dei quali contiene sempre le stesse 30 aziende fallite e 30 aziende sane randomizzate. Questo è stato fatto per evitare test-set "fortunati".

Modello	**Configurazione**	**Stopwords**	**Accuratezza (media di 20 test)**
Altman	-	-	0.75
LSTM	-	-	0.78
NLP	DistilBERT + Hadamard	SI	0.65
	DistilBERT + Hadamard	NO	0.70
	DistilBERT + Media	SI	0.81
	DistilBERT + Media	NO	0.73
	Doc2Vec	NO	0.71
Finale (NLP + LSTM)	DistilBERT + Hadamard	SI	0.83
	DistilBERT + Hadamard	NO	0.74
	DistilBERT + Media	SI	0.87
	DistilBERT + Media	NO	0.83
	Doc2Vec	NO	0.74

Tabella 7.2: Risultati di accuratezza sulla media di 20 test riportati per ogni modello e ogni configurazione.

Come si evince dalla tabella 7.2, il modello di ensemble finale ha performance migliori che i singoli moduli presi individualmente e del punteggio di Altman. Inoltre, notiamo che la configurazione migliore per la parte di NLP è quella che utilizza il dataset mantenendo le stopwords e che fa uso della media come tecnica di compattazione degli embedding. Di conseguenza, il modello finale migliore è quello appunto che come parte di NLP usa la medesima configurazione, con un'accuratezza dell'87%.

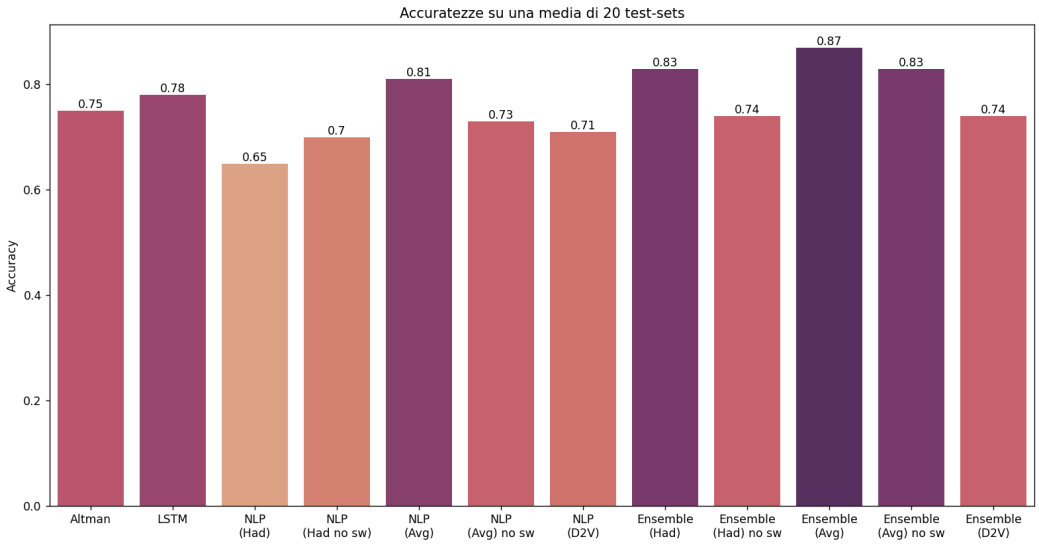

Figura 7.1: In figura è riportato un grafico a barre che riassume i risultati ottenuti.

L'87% di accuratezza è un risultato ottimo. Bisogna tenere conto, però, che le performance potrebbero variare qualora il modello sia testato su altri dati, è difficile infatti stimare la capacità di generalizzare di un modello come questo.

Conclusioni

L'obbiettivo prefissato per l'attività è stato raggiunto, si è progettato un modello composito addestrabile end-to-end che analizzi variabili finanziare e documenti 10-K con lo scopo di predire la bancarotta di un'azienda. Inoltre, si è svolta un'approfondita attività di confronto tra varie architetture del modello: i risultati ottenuti sui confronti sono di grande rilevanza per possibili aggiornamenti futuri. Oltretutto, come mostrato dalle performance riportate per il modello di *ensemble*, si sono ampiamente superati modelli di riferimento classici per quanto riguarda il task della predizione della bancarotta.

Limiti

Il modello proposto sicuramente presenta anche alcuni limiti:

- Il modello, soprattutto nella parte di NLP, dipende molto dalla struttura dei dati sui quali è stato addestrato. Se pur sia stato testato su una varietà di test-set, non è detto che con altri dati sia in grado di ottenere performance altrettanto valide. Se, per esempio, cambiasse anche di poco la struttura dei report 10-K, potrebbe non funzionare come dovuto.

- La parte di NLP basata su DistilBERT non prevede l'aggiornamento dei pesi di DistilBERT, in quanto si tratta di un'operazione computazionalmente troppo onerosa. Se si disponesse di macchine più performanti, in futuro si potrebbe effettuare un fine-tuning specifico per DistilBERT.

- Le tecniche utilizzate per unire i *chunk embeddings* hanno i loro limiti, già presentati in precedenza. Se pur siano state adottate delle tecniche che compensano questi limiti, potrebbero esistere tecniche più efficaci.

Sviluppi futuri

L'architettura del modello è stata progettata in maniera tale da permetterne l'estensione in maniera abbastanza agevole. Di seguito si riportano alcune idee per eventuali sviluppi futuri per migliorare il modello proposto:

- Si potrebbe aggiungere un modulo, oltre al modulo che analizza le variabili finanziarie e quello di NLP, che analizzi la collocazione di un'azienda all'interno di grafi finanziari, magari utilizzando *Graph Neural Networks*.

- Si potrebbe aggiungere un modulo di NLP che analizzi le news relative ad un'azienda, che spesso si rivelano di fondamentale importanza per capire la salubrità di un'azienda.

- Si potrebbe fare uno studio sull'efficacia di aggiungere ulteriori moduli, quindi determinare se esiste sempre un margine di miglioramento o se l'appesantire il modello con ulteriori moduli ne peggiora le prestazioni.

- Si potrebbe provare, in alternativa a DistilBERT classico, un DistilBERT specializzato su documenti finanziari, il quale potrebbe offrire performance migliori.

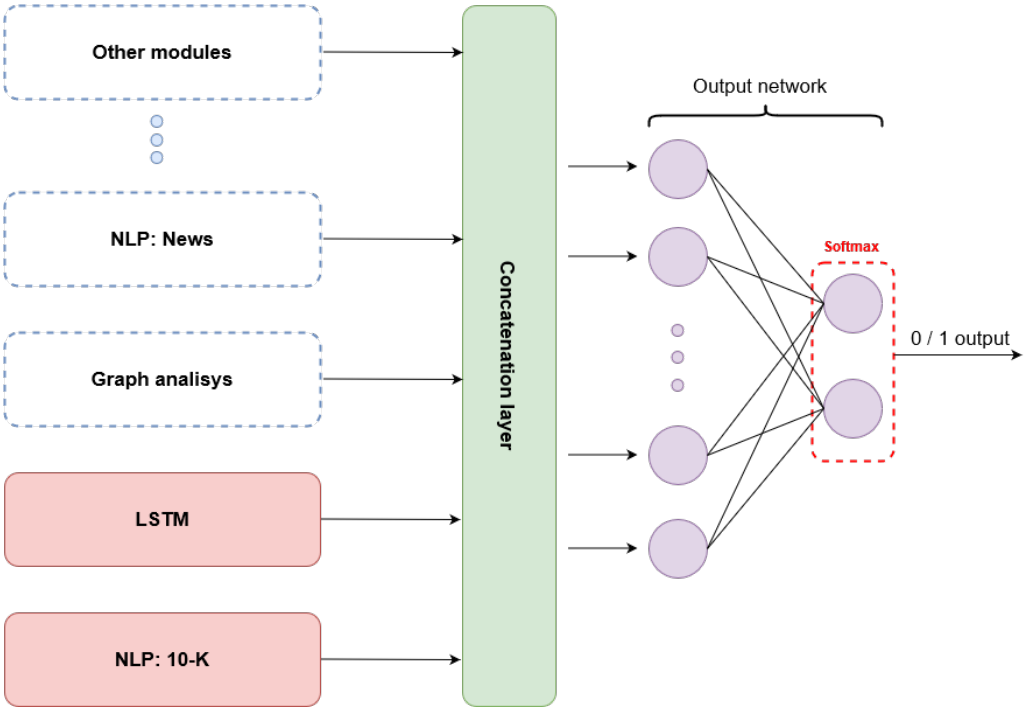

Figura 7.2: In figura è riportato uno schema che rappresenta un'eventuale architettura futura del modello.

Bibliografia

[1] Edward I. Altman. Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The Journal of Finance*, 23(4):589–609, 1968.

[2] Lauren Cohen, Christopher Malloy, and Quoc Nguyen. Lazy prices. *The Journal of Finance*, 75(3):1371–1415, 2020.

[3] Flavio Barboza, Herbert Kimura, and Edward Altman. Machine learning models and bankruptcy prediction. *Expert Systems with Applications*, 83:405–417, 2017.

[4] Marcus D Odom and Ramesh Sharda. A neural network model for bankruptcy prediction. In *1990 IJCNN International Joint Conference on neural networks*, pages 163–168. IEEE, 1990.

[5] Guoqiang Zhang, Michael Y Hu, B Eddy Patuwo, and Daniel C Indro. Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis. *European journal of operational research*, 116(1):16–32, 1999.

[6] Marek Vochozka, Jaromir Vrbka, and Petr Suler. Bankruptcy or success? the effective prediction of a company's financial development using lstm. *Sustainability*, 12(18):7529, 2020.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[12] Jay Alammar. The illustrated bert, elmo, and co. (how nlp cracked transfer learning), December 2018. jalammar.github.io.

[13] Joshi Prateek. How do transformers work in nlp? a guide to the latest state-of-the-art models, June 2019. analyticsvidhya.com.

[14] Abhilash Jain, Aku Rouhe, Stig-Arne Grönroos, and Mikko Kurimo. Finnish language modeling with deep transformer models. *CoRR*, abs/2003.11562, 2020.

[15] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[16] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[17] Remi Ouazan Reboul. Knowledge distillation for bert models: A theoretical and mechanistic study of the distilbert method, December 2021. zhuanlan.zhihu.com.

[18] Jay Alammar. A visual guide to using bert for the first time, November 2019. jalammar.github.io.

[19] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *CoRR*, abs/1908.10063, 2019.

[20] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[21] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[22] Sowide. Bankruptcy prediction dataset for american companies in the stock market. github.com/sowide.

[23] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[24] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[25] Tim Loughran and Bill McDonald. When is a liability not a liability? textual analysis, dictionaries, and 10-ks. *The Journal of finance*, 66(1):35–65, 2011.

Ringraziamenti

Un particolare ringraziamento è rivolto al Prof. Gianfranco Lombardo che, oltre ad essere stato sempre disponibile, mi ha dato l'opportunità di svolgere questa attività di tesi che si è rivelata molto interessante.
Ringrazio la mia famiglia per avermi permesso di portare a termine questi tre anni di studio.
Ringrazio Chiara perchè mi è sempre stata vicino anche nei momenti più difficili, ed è sempre presente qualora io abbia bisogno.
Un sentito ringraziamento va ai miei compagni di studio e amici Tombo e Bale che, fin dal primo giorno di università, hanno condiviso con me tempo, impegno e dedizione allo studio.
Ringrazio tutti i miei amici per avermi aiutato e supportato in maniera indiretta, in particolar modo Angelo, per essere stato un compagno fidato fin dalle scuole superiori, e Beatrice, per avermi sempre suppportato fin da quando ero piccolo.
Infine, posso porgere le congratulazioni anche a me stesso, per aver portato a termine questi 3 anni di sacrifici che mi hanno dato grandi soddisfazioni.