POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Classificazione testuale basata su summarization per fondi d'investimento

Relatore
prof. Paolo Garza
Correlatore
dott. Moreno La Quatra

Candidato
Federico Meneghetti

Anno Accademico 2022-2023

Sommario

I fondi comuni d'investimento consentono ai piccoli investitori di costruire portafogli diversificati. L'universo dei fondi d'investimento è molto vasto, date le molteplici variabili che li caratterizzano. Per questo motivo, un tema di particolare interesse è il loro inserimento in categorie, cioè insiemi che uniscono fondi con caratteristiche simili. Il progetto di tesi, sviluppato insieme all'azienda FIDA, è di costruire un classificatore basato su dati testuali che descrivono brevemente gli obiettivi e le finalità del fondo.

La metodologia applicata richiede l'utilizzo di tecnologie per il *Natural Language Processing (NLP)*, il settore del Machine Learning relativo all'analisi dei dati prodotti con linguaggio umano, tra cui i testi. Gli algoritmi per il NLP sono principalmente caratterizzati da architetture di reti neurali e lo stato dell'arte è costituito dai modelli *Transformer*. Nel lavoro, sono stati implementati algoritmi di classificazione testuale basati su Transformer e su *Doc2vec*, una rete neurale con un'architettura più semplice. Il metodo proposto consiste nell'effettuare uno step intermedio, preliminare alla classificazione. Sono stati infatti implementati algoritmi di text summarization, che consentono di generare riassunti a partire da un testo di input. L'idea è infatti di condensare le informazioni dei testi utili alla classificazione in testo più breve e con meno rumore. Anche in questo contesto sono state utilizzate architetture Transformer, con un addestramento su riassunti di riferimento costruiti sulla base di attributi in database.

In primo luogo, sono stati effettuati gli esperimenti di summarization, i cui risultati, misurati con le metriche *ROUGE* e *BERTScore*, si sono rivelati buoni. Successivamente, è stata implementata la classificazione testuale, mettendo a confronto i risultati ottenuti con i riassunti e quelli ottenuti con i testi originali. Si è notato che i riassunti hanno performance generalmente migliori, confermando la validità della soluzione proposta. Nonostante ciò, le metriche ottenute sono sicuramente migliorabili, lasciando perciò spazio a possibili sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Introduzione e contesto generale

Le analisi finanziarie hanno l'obiettivo di orientare gli investitori nelle proprie scelte. Una tipologia di strumento finanziario di particolare interesse per i piccoli investitori è costituita dalla vasta gamma dei fondi comuni d'investimento. In questo capitolo introduttivo si illustrano inizialmente i principi generali sulla costruzione di portafoglio. Vengono poi presentati i fondi d'investimento, descrivendo come essi vengono analizzati attraverso il processo sviluppato da FIDA. Infine, si illustra in che modo il Machine Learning, in particolare il Natural Language Processing, può essere utile in questo contesto.

1.1 Fondi comuni d'investimento

I fondi comuni di investimento sono un fondamentale strumento per costruire portafogli diversificati. Nei seguenti paragrafi si illustrano i principi base sulla costruzione di un portafoglio, per giustificare gli obiettivi e l'utilità dei fondi comuni d'investimento. Per questi concetti preliminari sono stati seguiti alcuni passaggi del capitolo 8 del libro di Brandimarte [4].

1.1.1 Investimenti e principio di diversificazione

Un investimento finanziario consiste nell'allocazione di capitale in una risorsa finanziaria, con la speranza di avere un ritorno economico maggiore di ciò che si otterrebbe non investendo il capitale, cioè conservandolo in un asset privo di rischio. Si denotano con $\tilde{r}$ e r_f i rendimenti dell'asset rischioso e di quello privo di rischio rispettivamente. Il rendimento dell'asset rischioso, in quanto incerto, è una variabile casuale per la quale si possono definire media e varianza come

$$\mathbb{E}[\tilde{r}] = \mu \quad \text{e} \quad \text{Var}(\tilde{r}) = \sigma^2.$$

L'asset rischioso, per essere preso in considerazione, deve necessariamente avere una speranza di rendimento maggiore rispetto al rendimento privo di rischio, deve cioè valere

$\mu > r_f$. L'asset rischioso presenta però un'imprevedibilità descritta dalla varianza σ^2. Un investitore, in base alla propria avversione al rischio, può decidere quanto capitale investire nell'asset rischioso e quanto in quello non rischioso, definendo quindi un portafoglio con rendimento

$$\tilde{r}_p(x) = x\tilde{r} + (1-x)r_f,$$

dove con x si indica la percentuale investita nell'asset rischioso. La media e la varianza del rendimento del portafoglio saranno quindi

$$\begin{aligned}
&\mathbb{E}[\tilde{r}_p(x)] = x\mu + (1-x)r_f = r_f + x(\mu - r_f),\\
&\mathrm{Var}(\tilde{r}_p(x)) = x^2\sigma^2.
\end{aligned}$$

Entrambe dipendono da x in maniera proporzionale, cioè maggiore è la quota investita nell'asset rischioso e maggiori sono media e varianza del portafoglio.

Per ridurre la varianza senza alterare il valore atteso si può ricorrere al *principio di diversificazione*, che consiste nella riduzione del rischio attraverso l'investimento di capitale in diversi asset. Si considerano n asset rischiosi $\tilde{r}_1, ..., \tilde{r}_n$ nei quali si suppone di investire in maniera equipesata. Se si indica con σ_i^2 la varianza dell'asset i e con σ_{ij} la covarianza tra asset i e asset j, si ottiene

$$\mathrm{Var}\left(\sum_{i=1}^{n}\frac{1}{n}\tilde{r}_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\sigma_{ij}.$$

Si può subito notare che la prima componente della varianza tende a zero per n che tende a infinito, mentre non vale per la seconda componente. Questo mostra la presenza di due tipi di rischio: un rischio *idiosincratico*, ovvero dipendente dal particolare asset ma eliminabile con la diversificazione, e un rischio *sistematico*, cioè non eliminabile.

Per ridurre il rischio eliminabile si può quindi far crescere il valore n, ovvero aumentare gli asset in cui investire, confermando così la validità del principio di diversificazione. Si è quindi giustificato matematicamente come gli investimenti più sicuri devono essere effettuati attraverso più risorse finanziarie, con un portafoglio diversificato. Si illustra nel seguito in che modo si può perseguire questo obiettivo nella pratica.

1.1.2 Costruzione di portafoglio e fondi d'investimento

La costruzione di un portafoglio finanziario è il processo di decisione nell'allocazione di capitale tra i diversi asset. Questo può essere fatto a partire dalla teoria moderna di portafoglio [23]. In particolare, si considera il problema di ottimizzazione di portafoglio in media-varianza:

$$\begin{aligned}
\max_{w_0, w}\quad & w_0 r_f + \mu^T w - \frac{\lambda}{2} w^T \Sigma w\\
\text{t.c.}\quad & w_0 + i^T w = 1
\end{aligned}$$

dove μ è il vettore dei valori attesi degli asset, Σ è la matrice di varianza e covarianza, w_0 è la frazione di portafoglio allocata nell'asset privo di rischio, w è il vettore delle frazioni

di capitale allocate nei diversi asset rischiosi e λ è il coefficiente di avversione al rischio. Si può dimostrare [4] che la soluzione del problema si realizza con

$$w^* = \frac{1}{\lambda}\Sigma^{-1}\pi, \quad w_0^* = 1 - \frac{1}{\lambda}\Sigma^{-1}\pi.$$

Nella pratica, tale soluzione non può essere applicata in maniera esatta, in quanto la natura degli asset finanziari è discreta (ad esempio, nel caso di un portafoglio azionario, ogni azione ha il proprio prezzo unitario). Questo significa che per costruire un portafoglio con le caratteristiche fornite dalla soluzione sarebbe necessario acquistare un consistente numero di azioni totali per rispettare le proporzioni tra i vari asset. In questo modo, l'investimento diventerebbe molto dispendioso per un piccolo investitore.

I fondi comuni di investimento nascono con lo scopo di consentire anche a chi ha una disponibilità economica limitata di investire in portafogli diversificati. Essi consistono nell'utilizzare le risorse economiche di molti individui per acquistare i titoli finanziari necessari a costruire un portafoglio adeguato. Se ne descrivono brevemente alcune caratteristiche [2].

Caratteristiche dei fondi

I fondi comuni di investimento sono gestiti dagli Organismi di Investimento Collettivo del Risparmio (OICR) che si distinguono in Fondi, SICaV e SICaF. Per i primi, il capitale raccolto è separato dal patrimonio della società che lo gestisce; per SICaV e SICaF invece, il capitale raccolto coincide con il patrimonio aziendale, gli investitori divengono quindi azionisti. Le SICaF, a differenza delle SICaV, sono OICR di tipo chiuso, cioè il capitale versato può essere restituito solo alla scadenza del fondo.

Vi sono molteplici fattori che entrano in gioco nel delineare l'universo dei fondi possibili. Essi possono essere quotati in borsa, oppure non quotati, possono essere aperti o chiusi (in base alla libertà sui tempi di sottoscrizione o rimborso); inoltre si differenziano in base alla politica di investimento, cioè l'insieme delle caratteristiche degli asset nei quali il fondo si impegna a investire. In particolare i fondi si possono distinguere sulla base della tipologia dei titoli che li compongono.

- Azionari: possono investire unicamente in azioni.
- Obbligazionari: possono investire unicamente in obbligazioni.
- Diversificati: possono investire in azioni e obbligazioni in proporzioni fissate a priori.
- Ritorno assoluto: possono investire in azioni e obbligazioni in proporzioni variabili nel tempo.
- Altri (monetari, materie prime,...).

Altre caratteristiche oggetto della politica di investimento sono l'area geografica in cui si vuole operare, la dimensione e i settori delle aziende in cui investire e l'esposizione valutaria.

Un'altra importante distinzione tra i fondi consiste nel tipo di gestione.

- Gestione attiva: i gestori del fondo cercano di ottenere risultati migliori del mercato, utilizzando una strategia dinamica per la compravendita dei titoli.

- Gestione passiva: per questa tipologia di fondi, l'obiettivo è semplicemente di replicare l'andamento medio del mercato, seguendo il comportamento di un certo indice di mercato. I titoli vengono quindi acquistati in modo da riprodurre la composizione dell'indice scelto, per questo sono più semplici da gestire.

Un esempio di particolare rilevanza è quello degli ETF (Exchange-Traded Fund), con cui si denotano fondi o SICaV a gestione passiva quotati in borsa. Date le loro caratteristiche, essi hanno tipicamente basse spese di gestione.

In conclusione, si può notare che fattori che concorrono a delineare le caratteristiche di un fondo d'investimento sono molteplici. L'universo dei fondi è perciò molto ampio e diversificato e individuare le caratteristiche che rendono un fondo più o meno adatto a un certo individuo è un'operazione complessa.

Documentazione

Le caratteristiche di un fondo comune d'investimento precedentemente riportate possono essere estratte da diversi documenti che la società di emissione è tenuta a redigere. In primis vi è il prospetto informativo, un documento che riporta l'intero regolamento e le caratteristiche del fondo. Tutte le informazioni sul fondo devono essere riportate nel prospetto. Il KIID (Key Investor Information Document) ha invece lo scopo di riassumere in maniera sintetica le caratteristiche principali del fondo, in modo le informazioni possano essere facilmente comprese dall'investitore. Vi è infine il KID (Key Information Document), una versione del KIID con un formato standardizzato.

1.2 Il sistema FIDArating

Data la complessità e la varietà del mondo dei fondi di investimento, le aziende specializzate in analisi finanziarie hanno sviluppato sistemi per effettuare classificazioni e analisi quantitative. FIDArating [9] è il processo sviluppato dalla società FIDA che, sulla base di fattori quantitativi e qualitativi, ha l'obiettivo di analizzare strumenti finanziari. Esso si divide in tre fasi:

- Classificazione

- Categorizzazione

- Rating

Questo processo è del tutto generale e può essere applicato all'analisi di qualsiasi tipo di strumento finanziario. Ai fini della trattazione, l'interesse è però volto all'analisi dei fondi d'investimento. Nel seguito, si illustra il significato di queste tre fasi, descrivendo le operazioni che le caratterizzano.

1.2.1 Classificazione

Il processo di classificazione consiste nell'analisi delle caratteristiche di un particolare strumento finanziario, descrivendolo in maniera schematica attraverso una serie di attributi. Essi sono riassunti nella tabella seguente.

N°	**Campo**	**Contenuto**	**Descrizione**
1	ASSET	Asset class	Asset class di riferimento dello strumento
2	GEO_AREA	Area geografica	Macro-area geografica di esposizione
3	GEO_ZONE_COUNTRY	Paese o zona geografica	Il paese specifico o la zona geografica limitata all'interno della macro-area
4	SECTOR	Macrosettore	Settore generico a cui è esposto lo strumento
5	SUB_SECTOR	Settore	Settore specifico all'interno del macrsettore a cui è esposto lo strumento
6	DIMENSION	Dimensione	Dimensione/i della/e società a cui lo strumento fa riferimento. La dimensione è contestualizzata al mercato di riferimento
7	STYLE	Stile di investimento	Indica la natura delle società dal punto di vista della nota metodologica dello stile di investimento (Growth - Value)
8	RATING	Classe di Rischio	Solidità patrimoniale in coerenza con gli standard adottati dalle principali agenzie di rating internazionali
9	MATURITY	Duration	Duration/Scadenza di riferimento
10	CURRENCY_EXPOSURE	Esposizione alle valute	Esposizione valutaria
11	ASSET_EXPOSURE	Esposizione verso le asset class	Percentuale di esposizione alle principali asset class
12	TYPE	Tipologia	A seconda del tipo di strumento ne sintetizza alcune caratteristiche peculiari (ad es. per i fondi il tipo di strategia)

13	ISSUER	Emittente	Tipologia di emittente dello strumento

Tabella 1.1: Tabella classificazione.

Per quanto riguarda i fondi d'investimento, l'analisi si riferisce agli strumenti che compongono il portafoglio del fondo nella sua continuità temporale, cioè alla tipologia di titoli in cui il fondo può investire, descritta dalla politica d'investimento. Queste informazioni vengono dedotte sia dalle dichiarazioni di intenti (analisi dei prospetti) sia dalle realizzazioni effettive (analisi del portafoglio).

Analisi dei prospetti

In primo luogo, si analizza l'insieme delle regole dichiarate dalla società in merito alla gestione del fondo, vale a dire la politica d'investimento. Come riportato precedentemente, queste informazioni sono reperibili nella documentazione legale che la società emittente deve rilasciare.

Altre informazioni possono invece derivare dalla dichiarazione del benchmark dell'investimento, vale a dire un parametro di rifermento che si mira a raggiungere o superare. Esso non è necessariamente legato alla natura degli investimenti del fondo.

Analisi del portafoglio

In aggiunta alle dichiarazioni di intenti da parte dei gestori dei fondi, si può studiare l'effettiva composizione di portafoglio nel tempo, grazie alle pubblicazioni periodiche da parte delle società. Oltre a ciò, è possibile utilizzare delle metodologie quantitative che, a partire dalle realizzazioni puntuali di portafoglio, desumano lo stile di investimento, confrontandolo con gli indici di mercato.

1.2.2 Categorizzazione

Il secondo step consiste nell'inserire ogni fondo in una specifica *categoria*, cioè un insieme di strumenti omogenei tra loro, sulla base delle caratteristiche della Tabella 1.1. Questa operazione viene fatta con uno sistema gerarchico a più livelli di dettaglio, costituito da:

- Superclasse
- Classe
- Categoria

Il livello più specifico è rappresentato dalle categorie. Un esempio di categoria può essere *Azionari America Latina Big Cap* oppure *Azionari Europa - Mid Cap EUR*. Raggruppando i fondi secondo un livello si dettaglio inferiore si ottengono le *classi* che solitamente riuniscono gli strumenti finanziari con la stessa area geografica e la stesso tipo di

asset. Infine, il livello gerarchico più elevato è costituito dalle *superclassi*, che raggruppano strumenti sulla base dell'asset class.

Lo scopo delle operazioni svolte fino a questo punto non è solo di schematizzare l'universo dei fondi, ma anche di raggrupparli in categorie per poter fare analisi quantitative sulla base della categoria di appartenenza. Come operazione preliminare, viene individuato un benchmark di categoria attraverso analisi statistiche che mostrano quale sia l'indice di mercato più appropriato per una data categoria. Nei casi in cui questo indice non viene individuato, il benchmark è costituito da una media pesata di due o più indici di mercato.

1.2.3 Rating

Il processo di rating consiste nell'attribuire un punteggio di merito per ciascun fondo, considerando il rispettivo universo di riferimento (la categoria di appartenenza). Per ogni categoria, i fondi vengono poi ordinati e suddivisi in 5 percentili, in modo da attribuire una classe di merito.

Senza entrare nel dettaglio di come il calcolo del rating viene effettuato, si riporta soltanto che esso dipende da misure legate all'andamento del fondo, del benchmark di categoria e dell'asset privo di rischio. Inoltre, affinché venga calcolato il rating, vi sono requisiti che devono essere soddisfatti dal fondo.

1.3 Natural Language Processing per FIDA

Il sistema FIDArating ha come punto di partenza lo studio della documentazione relativa al fondo d'investimento e tutte le operazioni successive devono essere svolte manualmente. Questo significa che è necessario il lavoro di un esperto del settore che si possa dedicare alle seguenti mansioni:

- estrazione di informazioni su politica e finalità dell'investimento a partire della documentazione (prospetto e KIID);
- schematizzazione attraverso classificazione e categorizzazione.

L'obiettivo di queste operazioni è ricondurre i fondi alle categorie sulle quali fare analisi e attribuire livelli di rating, oltre ad avere una base dati standardizzata che racchiude le principali caratteristiche dei fondi. Queste attività sono onerose in termini di tempo e un obiettivo aziendale è indagare su strumenti che automatizzino il processo, o parti di esso. In questo elaborato si è interessati, in particolare, allo sviluppo di sistemi di Machine Learning che automatizzino il processo di categorizzazione, che si traduce nella costruzione di algoritmi di classificazione.

Un primo esperimento sviluppato da FIDA consiste nell'effettuare una classificazione sulla base delle serie storiche dei rendimenti dei fondi. Ci si è però scontrati con il problema che alcuni fondi, seppur diversi tra loro, non siano distinguibili sulla base dei soli rendimenti. Per questo motivo si è spostato l'interesse su altri dati di input che potessero spiegare meglio le differenza tra i fondi, cioè i dati testuali. In database sono infatti presenti campi testuali costituiti da nome del fondo, politica di investimento e finalità del

fondo, che verranno illustrati meglio nel Capitolo 3. Gli ultimi due sono estratti di testo presente nella documentazione ufficiale e contengono le informazioni principali dalle quali è possibile redigere la tabella di classificazione descritta nella sezione 2.3. L'idea sviluppata consiste quindi nel costruire algoritmi di classificazione che utilizzino questi campi testuali per associare a ogni fondo la rispettiva categoria. Oltre all'obiettivo di classificazione ne è stato sviluppato un altro collaterale che si integra con il primo, ovvero la *Text Summarization*, il riassunto automatico di dati testuali. La disciplina che si occupa degli algoritmi di Intelligenza Artificiale nel contesto di dati testuali è detta Natural Language Processing (NLP). Le tecnologie consolidate in ambito NLP sono basate su architetture di reti neurali artificiali.

Il Capitolo 2 è dedicato alla trattazione teorica degli strumenti di NLP utilizzati nel lavoro di tesi, descrivendone i principi di base e le architetture utilizzate, fino ad arrivare allo stato dell'arte in ambito NLP: l'architettura *Transformer*. Nel Capitolo 3 verrà invece presentato il dataset aziendale utilizzato per gli esperimenti, con le relative operazioni di preprocessing. Si passerà poi, nel Capitolo 4, all'esposizione della metodologia proposta per la classificazione dei fondi e alla descrizione degli specifici modelli utilizzati nel lavoro. In seguito, saranno illustrate le metriche per la valutazione dei modelli e le configurazioni di parametri utilizzate negli esperimenti. Questi argomenti, insieme ai risultati ottenuti, sono l'oggetto del Capitolo 5. Infine, nel Capitolo 6 si riporteranno le conclusioni e i possibili sviluppi futuri.

Capitolo 2

Natural Language Processing

Il Natural Language Processing (NLP) è l'insieme di metodi per rendere il linguaggio umano accessibile ai computer [8]. Lo sviluppo di questa disciplina è motivato sia dal proposito di sviluppare sistemi che consentano l'interazione uomo-macchina attraverso il linguaggio naturale, sia dall'esigenza di estrarre valore dalla grandissima quantità di dati testuali presenti nel mondo. Lo scopo di questo capitolo è di illustrare i principali concetti e architetture utilizzati nel lavoro di tesi. In primo luogo, verrà illustrato in che modo i dati testuali possono essere matematicamente rappresentati attraverso vettori ad alte dimensioni. Si spiegherà come questo può essere fatto grazie alle architetture *Word2vec* e *Doc2vec*. Si presenterà poi la più importante architettura in ambito NLP, i *Transformer*, descrivendone la struttura e i meccanismi principali. Infine, verrà mostrato in che modo le tecnologie descritte possono essere utilizzate per effettuare text classification e text summarization.

2.1 Modelli di Embedding

In questa sezione si descrive in che modo i dati testuali possono essere rappresentati mediante vettori numerici attraverso le architetture Word2vec e Doc2vec. Per questa tipologia di vettorizzazione testuale, è comune applicare degli step di preprocessing, che vengono descritti di seguito.

2.1.1 Preprocessing dei testi

Il preprocessing è l'insieme di operazioni di pulizia e standardizzazione dei dati, per renderli utilizzabili dai modelli di machine learning. Nel caso di dati testuali, il preprocessing consiste nella creazione e nella standardizzazione delle unità testuali (token).

La operazioni descritte si riferiscono al contesto delle rappresentazioni vettoriali di dati testuali (gli embedding) basati su tecnologie Word2vec e Doc2vec. Lo stato dell'arte per gli embedding testuali è però rappresentato dall'architettura Transformer, per la quale il preprocessing viene eseguito diversamente, come mostrato nella sezione (2.2.1).

Pulizia testo e stopword

La primissima fase riguarda la pulizia del testo, vale a dire la rimozione o la modifica di tutti gli elementi considerati inutili o fastidiosi. Questi elementi possono essere caratteri speciali, segni di punteggiatura, o specifici elementi in un determinato contesto. In secondo luogo si procede ad eliminare le parole più frequentemente utilizzate in una lingua, dette *stopword*. Queste parole includono articoli, preposizioni e parole di uso così comune tali da non apportare informazioni al testo. Esse vengono perciò eliminate in modo da avere un testo che contenga solamente le parole semanticamente rilevanti.

Tokenizzazione

La fase successiva consiste nello scomporre il testo in una sequenza di unità fondamentali, dette *token*. I token possono essere frasi, insiemi di parole, singole parole o porzioni di esse. Un tokenizer è una funzione che riceve una stringa testuale e restituisce la lista dei token. Questa operazione è ciò che consente di creare un set di dati standardizzati e processabili dal modello.

Nel caso in cui come token si utilizzino le singole parole, può essere effettuata la lemmatizzazione, un'operazione che consiste nel ricondurre ogni parola (token) al proprio *lemma* (ad esempio: mangio -> mangiare, alberi -> albero). Un lemmatizer è una funzione che esegue questa operazione su ogni parola. Si possono però verificare situazioni con conflitti, cioè possono esserci parole con più di una possibile lemmatizzazione (es. accordo -> accordo, accordo -> accordare). Per questo motivo, molti lemmatizer individuano la parte del discorso relativa a ogni parola del testo (come verbo, aggettivo, ecc.), in modo da assegnare il lemma nella maniera più corretta, riuscendo a compiere la giusta distinzione nei casi ambigui.

Un'ulteriore operazione consiste nel ricondurre ogni parola alla propria radice. La motivazione è che le parole con la stessa radice sono semanticamente molto simili. Ogni parola viene quindi troncata (es calcolare -> calcol, calcolo -> calcol), in questo modo, diverse parole vengono ricondotte alla stessa radice, riducendo la variabilità dei dati (cioè il numero di token diversi) senza ridurre quella semantica. Questo passaggio è definito *stemming*, e la funzione che lo esegue è detta stemmer.

2.1.2 Embedding

I procedimenti illustrati fino a questo punto hanno trasformato un testo in una collezione di elementi standardizzati. Questi dati necessitano però di una rappresentazione matematica, per poter essere essere trattati dagli algoritmi, come avviene per un qualsiasi altro tipo di dato (numerico, categorico, ecc..). Questo significa che è necessario costruire dei vettori che rappresentino nel miglior modo possibile il significato semantico del testo di partenza. La rappresentazione vettoriale degli elementi testuali è detta *embedding*. Questa operazione può essere svolta con metodi molto semplici, come *Bag-of-Words (BoW)* o *Term Frequency-Inverse Document Frequecy (TF-IDF)*, oppure in maniera più elaborata usando particolari architetture di reti neurali come Word2vec o Doc2vec, fino ad arrivare alle più complesse architetture Transformer.

L'obiettivo è rappresentare una serie di documenti con dei vettori numerici, uno per ogni documento. Sia con il BoW che con il TF-IDF ogni documento costituisce una riga di una tabella, mentre ogni colonna rappresenta una parola del dizionario di parole che costituiscono l'insieme di documenti.

Il Bag-of-Words consiste nel contare il numero di occorrenze di una parola all'interno di un determinato documento e nel riportarlo nella corrispondente entrata della tabella. Si può notare che con questa strategia, le parole molto frequenti in tutti i documenti verranno ritenute importanti per ogni documento.

Il TF-IDF consiste invece nell'attribuire alla parola w nel documento d il peso

$$\text{TF-IDF}(w, d) = \frac{n(w, d)}{|dim(d)|} \cdot \log_{10} \frac{|D|}{|d : i \in D|},$$

dove $n(w, d)$ è il numero di volte che la parola w compare in d, $|dim(d)|$ è il numero di parole in d e $|D|$ è il numero di documenti. In questo modo si dà peso alle parole che sono molto presenti in uno specifico documento, ma meno comuni in tutti gli altri, ritenendole parole di maggiore rilevanza (cioè che caratterizzano un particolare testo).

Questi semplici metodi di vettorizzazione consentono di ottenere un vettore numerico, di dimensione pari al numero di parole del dizionario che costituisce i testi, per rappresentare un testo. Il principale svantaggio di questi approcci è la dimensione molto elevata degli embedding generati e la conseguente naturale tendenza ad avere vettori molto sparsi. Un altro punto debole è il fatto che non si colgono le informazioni legate alla posizione delle parole, cioè due testi con le stesse parole, ma ordine delle parole diverso, hanno la stessa rappresentazione vettoriale. Si illustrano perciò tecnologie più avanzate per costruire embedding.

2.1.3 Word2vec

Un obiettivo degli embedding testuali è di trasformare le singole parole in oggetti matematicamente trattabili, effettuando cioè un *word embedding*. L'idea alla base del word embedding è di rappresentare le parole come punti di uno spazio vettoriale di grandi dimensioni, in maniera che la loro posizione rappresenti il loro significato semantico. Significa che parole con significati simili devono essere rappresentate con vettori tra loro simili, mentre parole con significati molto diversi attraverso vettori molto lontani tra loro.

Word2vec è un modello di rete neurale che, processando un testo, costruisce embedding delle parole che lo compongono. Il funzionamento di Word2vec è basato sull'ipotesi fondante della semantica distribuzionale, ovvero che il significato semantico di ogni parola sia dettato dal contesto in cui essa si trova, cioè dall'insieme di parole che stanno attorno ad essa [18]. Perciò due parole sono tanto più simili quanto più tendono a comparire nello stesso contesto.

Per descrivere il modello Word2vec si fa riferimento al paper originale [24] pubblicato nel 2013 e a [17]. Ci si riferirà alle unità testuali con il termine *parole*, da momento che, con questa architettura, la tokenizzazione tipicamente utilizzata consiste nella suddivisione per parole. Il modello funziona grazie all'addestramento di una rete neurale, i cui pesi finali costituiscono la rappresentazione delle parole. Vi sono due architetture possibili, il *Continuous Bag-of-Words (CBOW)* e lo *Skip-gram*. La prima addestra la rete cercando

di prevedere le parole corrente a partire dalle parole attorno ad essa, mentre la seconda cerca di prevedere le parole vicine data la parola corrente. In entrambi i casi, le operazioni vengono effettuate su una finestra di contesto di n parole, cioè si considerano le $n-1$ parole attorno alla parola in esame. L'architettura è costituita da una rete poco profonda, con layer di input e di output un solo layer nascosto.

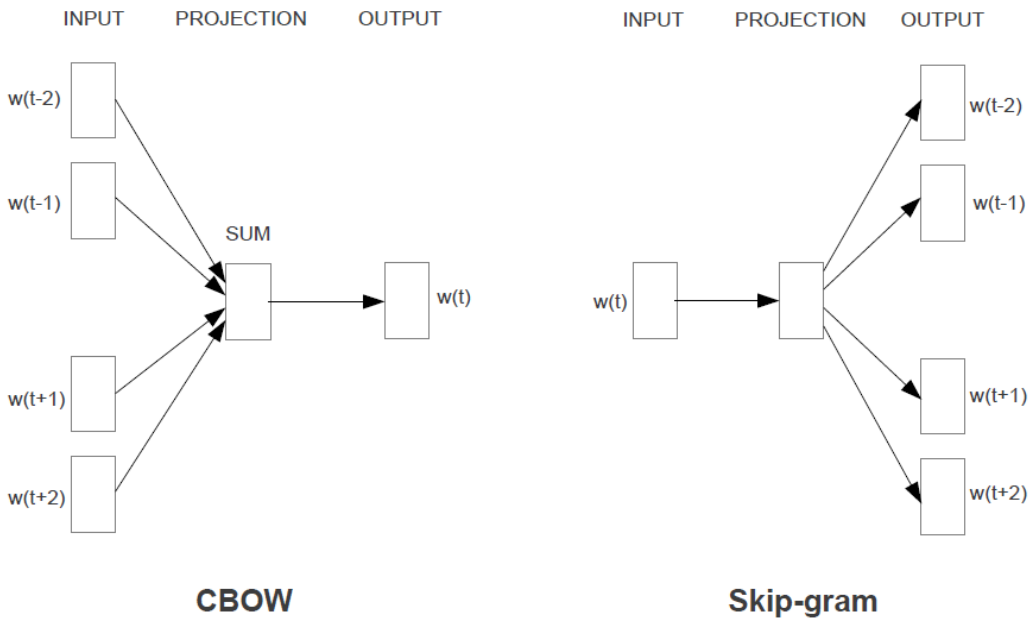


Figura 2.1. Schema delle 2 possibili architetture per il modello Word2vec riportato sul paper originale [24].

Skip-gram

Uno Skip-gram è un n-gramma, cioè un insieme di $n-1$ parole che presenta una parola mancante al centro di esso. Uno Skip-gram con n pari a 5 per una data parola sarà perciò costituito dalle 2 parole precedenti e dalle 2 successive alla parola data. L'input della rete neurale è il vettore one-hot encoding per la parola a cui si riferisce lo Skip-gram (che è la parola da cui devono essere inferite le altre parole di contesto). Per ogni parola, vengono create delle coppie di training costituite dalla parola di input e una parola dello Skip-gram. L'output della rete neurale è una distribuzione di probabilità sull'insieme delle parole della frase. L'obiettivo è ottimizzare i parametri per predire la parola dello Skip-gram relativa alla coppia di training, concentrando cioè la probabilità dell'output sulla parola di contesto che deve essere individuata. Questa operazione viene effettuata iterativamente su tutte le coppie di training per lo Skip-gram attuale e su tutti gli Skip-gram del testo. Alla fine dell'addestramento, i vettori di embedding delle parole di ottengono a partire dalla

matrice di pesi del layer nascosto. Ogni riga della matrice di pesi costituisce l'embedding della parola corrispondente a tale riga. In altre parole, basta moltiplicare la matrice dei pesi per il vettore one-hot encoding della parola di interesse per ottenere il suo embedding.

CBOW

Il funzionamente del Continuous Bag-of-Words è simile all'architettura Skip-gram, con la differenza che il vettore di input è dato dalla somma dei vettori one-hot encoding dei vettori di contesto. Infatti, a partire da questi, la rete viene addestrata per individuare la parola al centro del contesto. Come per l'architettura Skip-gram, si itera l'addestramento su tutti gli n-grammi mascherati del testo in esame. Anche in questo caso, i vettori di embedding sono costituiti dalle righe della matrice dei coefficienti.

2.1.4 Doc2vec

Le architetture del modello Word2vec consentono di ottenere rappresentazioni semantiche per ogni singola parola. L'interesse più generale è però di ottenere una rappresentazione vettoriale di un'intera sequenza testuale. Il modo più semplice in cui ciò può essere fatto è calcolando la media aritmetica dei vettori Word2vec di tutte le parole di un testo. Esiste però un'architettura che esegue questa operazione in maniera più sofisticata: il Paragraph Vector, anche detto Doc2vec, presentato nel 2014 [25]. Questo modello addestra una rete neurale in maniera simile al Word2vec, ma includendo anche un vettore che rappresenta l'intero testo. Esso viene aggiornato durante la scansione dei token, che anche in questo caso corrispondono alle singole parole. Vi sono due possibili architetture per la rete neurale, la *Distributed Memory (DM)* e la *Distributed Bag-of-Words(DBOW)*, che sono i corrispettivi per il paragraph vector embedding di CBOW e Skip-Gram rispettivamente.

Distributed Memory

La differenza tra questa architettura e la CBOW è che, oltre alle parole di contesto, viene dato in input anche un vettore di one-hot encoding per il testo di riferimento nella lista dei testi presi in considerazione. In questo modo, oltre alla matrice di pesi $\boldsymbol{W}$ relativa alle parole di contesto, è presente un'altra matrice $\boldsymbol{D}$ che codifica l'intero testo (paragrafo) considerato. Ogni riga della matrice $\boldsymbol{D}$ costituirà quindi l'embedding per il testo a cui corrisponde. Il training avviene eseguendo iterazioni su tutte le finestre di contesto del testo, includendo sempre il vettore relativo al paragrafo. L'idea di questa architettura è infatti di costruire una matrice che preservi l'informazione semantica di tutte le parole del testo in considerazione. Questa operazione viene poi iterata su tutto il corpus di testi in esame, ottenendo la matrice finale $\boldsymbol{D}$ che conterrà le rappresentazioni vettoriali di tutti i testi.

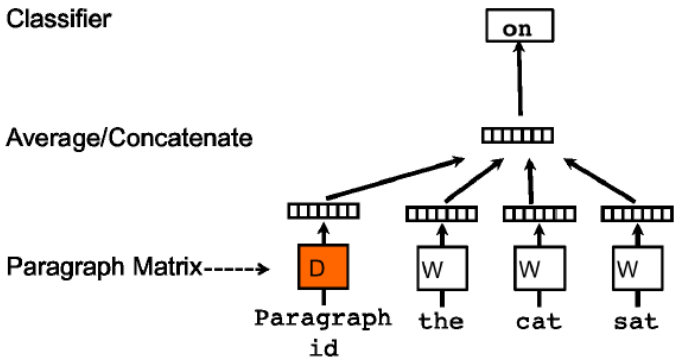


Figura 2.2. Rappresentazione dell'architettura Distributed Memory [25].

Distributed Bag-of-Words

Questo secondo tipo di architettura consiste nel ricevere in input solamente il vettore one-hot encoding del paragrafo considerato e nel cercare di prevedere l'output, che consiste in una parola campionata dal paragrafo. In particolare viene effettuato un campionamento per quanto riguarda la finestra di contesto e poi un successivo campionamento sulla parola della finestra di contesto da prevedere. I pesi del modello vengono quindi ottimizzati di modo che dal paragrafo si riescano a prevedere le parole che lo compongono. In questo modo, la memoria richiesta si riduce notevolmente, in quando bisogna memorizzare solamente i pesi della matrice D. Inoltre il training risulta anche più rapido e agevole. Di conseguenza, però, il modello non tiene conto dell'ordine delle parole, in quanto esse vengono campionate casualmente e per questo motivo l'accuratezza è generalmente inferiore rispetto a quella dell'architettura Distributed Memory.

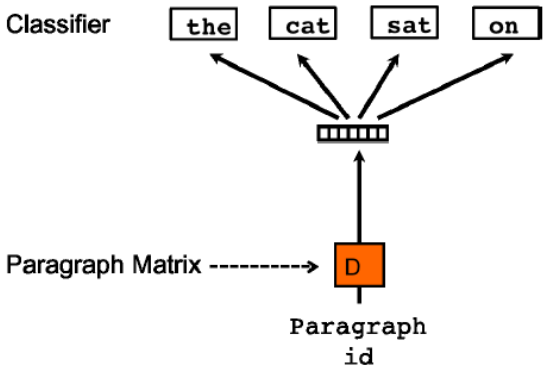


Figura 2.3. Rappresentazione dell'architettura Distributed Bag-of-Words [25].

2.2 Transformer

L'interesse per il Natural Language Processing ha permesso un crescente sviluppo di architetture di reti neurali sempre più complesse, tra cui spiccano le Recurrent Neural Networks (RNN) [35], fino ad arrivare alla tecnologia che si affermata negli ultimi anni: l'architettura Transformer. Il meccanismo fondamentale su cui è basata questa architettura è detto *Attention*, come si evince dal titolo del paper originale *Attention is all you need* [38], pubblicato nel 2017. L'architettura Transformer consente di eseguire operazioni molto diverse tra loro in ambito NLP, a differenza delle architetture Doc2vec e Word2vec che sono progettate per lo specifico compito di produrre vettori di embedding. Lo scopo di questa sezione è di illustrare il funzionamento dell'architettura Transformer, facendo riferimento sia al paper originale, sia al libro dedicato all'utilizzo di questi strumenti attraverso la libreria distribuita da HuggingFace [36].

2.2.1 Preprocessing e subword token

A differenza dei modelli di embedding come Doc2vec, il preprocessing per l'architettura Transformer non prevede operazioni come eliminazione stopword e stemming, dato che la complessità del modello consente di catturare relazioni logiche e semantiche tra tutte le parole utilizzate. Per gestire la dimensione del dizionario utilizzato è comune ricorrere all'utilizzo di tokenizzatori basati su porzioni di parole (*subword*). Come riportato in [36], il processo di creazione di token può essere diviso in quattro fasi.

- *Normalizzazione*. Questa fase può includere diverse operazioni di pulizia del testo, come l'eliminazione dei caratteri accentati e la trasformazione del testo in caratteri minuscoli. Può anche essere effettuata la *normalizzazione Unicode* [40], che gestisce le situazioni in cui sono presenti codifiche diverse per lo stesso carattere,

- *Pre-tokenizzazione*. Essa consiste nella suddivisione del testo nelle sue naturali unità testuali, costituite dalle singole parole e dai segni di punteggiatura.

- La *tokenizzazione* vera e propria consiste nell'applicazione di algoritmi per la creazione dei subword token, che scompongono le parole in sotto parole. I più comuni sono WordPiece [42], Byte Pair Encoding [33] e Unigram [16]. L'idea generale di questi algoritmi (descritti alla sezione (3.3.4)) è di scomporre le parole nelle sotto parole più comuni del testo, in modo da avere un numero di subword totali non troppo elevato. Le parole più comuni vengono solitamente lasciate intere.

- *Postprocessing*. In questa ultima fase vengono aggiunti dei token speciali, come [CLS] (a inizio frase) o [SEP] (a fine frase).

Il principale vantaggio di questo tipo di tokenizzazione rispetto a quella basata su parole è di riuscire a ottenere un vocabolario di token non troppo grande senza avere perdite ortografiche o lessicali. Anche in questo caso a ogni token viene associato un numero (id), che a sua volta viene trasformato in una rappresentazione one-hot encoding.

Si mostra un esempio di tokenizzazione WordPiece, riportato nel capitolo 2 di [36]. La frase

'Tokenizing text is a core task of NLP'

viene tokenizzata nel seguente modo:

['[CLS]', 'token', '##izing', 'text', 'is', 'a', 'core', 'task', 'of', 'nl', '##p', '.', '[SEP]'].

Si può notare che i token costituiti da pezzi di parole diversi da prefissi, cioè quelli il cui token precedente costituisce un pezzo della stessa parola, vengono stampati con il prefisso ##. Oltre al trasposizione del testo in lettere minuscolo sono presenti anche i token speciali [CLS] e [SEP].

Quando viene effettuata la tokenizzazione su una serie di documenti si possono specificare diversi parametri, in particolare vi sono:

- *padding*, che rende tutte le sequenza lunghe uguali, aggiungendo dei token speciali (denotati con [PAD]) per di modo da raggiungere la massima lunghezza prestabilita;

- *truncation*, che consente di troncare il testo in input ad una lunghezza specificata o alla lunghezza massima ammissibile per il modello.

Quando una lista di testi viene tokenizzata, per ogni testo vengono creati due attributi. Il primo, denotato con *input_ids*, è la lista degli id corrispondenti ai token del testo. Il secondo elemento è detto *attention_mask* ed è una lista di lunghezza pari al numero di token e serve a specificare se il token nella posizione corrispondente deve essere considerato (se attention_mask = 1 in quella posizione) o se non va considerato perché è un token di padding (in questo caso vale 0).

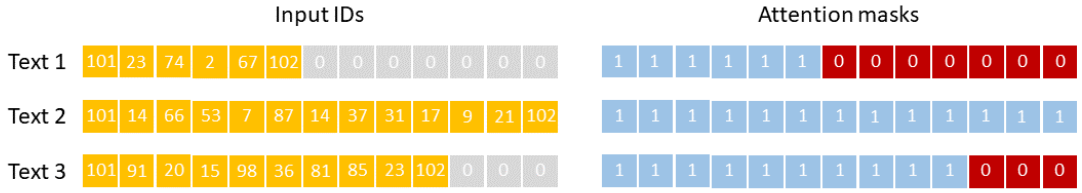


Figura 2.4. Ogni sequenza testuale viene rappresentata con la stessa lunghezza, riempiendo i token mancanti con il token [PAD] che ha id pari a 0. L'attention mask segnala al modello quando il token va considerato e quando è un token di padding [36].

Dopo questo processo di codifica, i one-hot encoding dei token vengono mappati in uno spazio vettoriale di dimensione minore, attraverso parametri addestrabili, costruendo così gli embedding del testo. A questo punto i testi hanno completato la loro fase di preprocessing e possono essere utilizzati dal modello.

2.2.2 Architettura Transformer

L'architettura Transformer, nata nel 2017, ha rivoluzionato il mondo del Natural Language Processing, superando i limiti delle reti neurali ricorrenti (RNN). Il paper originale [38] presenta l'architettura per un task di traduzione testuale e la struttura dell'architettura è costituita da due elementi: *Encoder* e *Decoder*. Questo tipo di architettura è anche detto *sequence-to-sequence* in quanto consente di trasformare una sequenza testuale in un altra sequenza, come può avvenire ad esempio per una traduzione, che è il task presentato nel paper. L'encoder ha l'obiettivo di codificare il testo in un embedding, anche detto *stato nascosto*, mentre il decoder serve a generare i token di output a partire dallo stato nascosto.

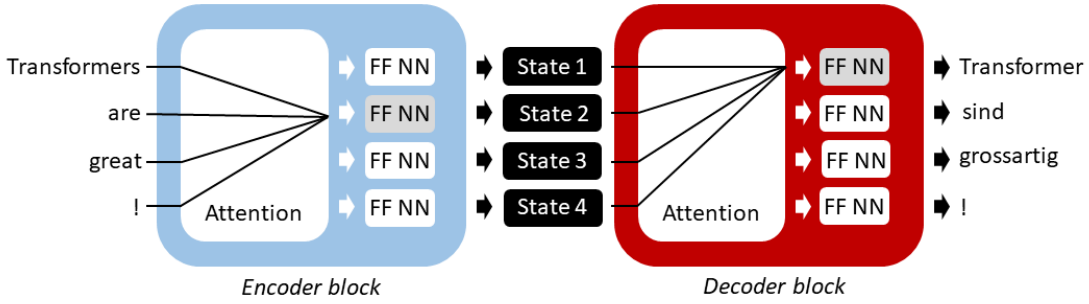


Figura 2.5. Schema generale del funzionamento dell'architettura Transformer per la traduzione automatica, in riferimento all'esempio in [36].

Come anticipato, il meccanismo principale dei Transformer è l'*attention*, in una sua particolare forma detta *self-attention*. Se si considera una sequenza testuale, codificata come un insieme di token, l'attention è un peso che si associa a ogni token della sequenza per denotare la sua rilevanza rispetto a un altro token. Nei Transformer, questa procedura viene applicata sui token della sequenza stessa, cioè ogni token ha un coefficiente che indica la propria importanza rispetto a ogni altro token della sequenza. Per ogni token della sequenza, gli embedding di tutte le altre parole vengono poi combinati con una media pesata sulla base dell'attenzione. Queste combinazioni vengono successivamente passate a delle reti neurali feed-forward, ognuna delle quali produce l'output detto *stato nascosto*. Questi output vengono poi dati in input al decoder che, utilizzando self-attention e reti feed-forward, producono l'output sotto forma di testo.

Oltre all'architettura Encoder-Decoder, in cui l'operazione constiste nel passare da un testo di input a un altro testo di output, encoder e decoder possono esistere come componenti singole. Un'architettura composta dal solo encoder ha come output gli embedding testuali e può essere utile per eseguire operazioni come Text Classification. Le architetture composte dal solo decoder possono essere invece utili nel completamento automatico di frasi.

La schema in Figura 2.5 rappresenta la struttura di un Transformer in maniera molto semplificata. Prima di analizzare più nel dettaglio le varie componenti dell'architettura si riporta una rappresentazione più dettagliata, tratta dal paper originale [38], dalla quale si possono subito notare le due principali componenti, presenti sia dell'encoder che nel decoder: La Multi-Head Attention e le reti Feed Forward. Si può inoltre notare come l'intera architettura dell'encoder, così come quella del decoder, costituisce un solo blocco del Transfomer e che ognuno viene applicato iterativamente un numero N di volte.

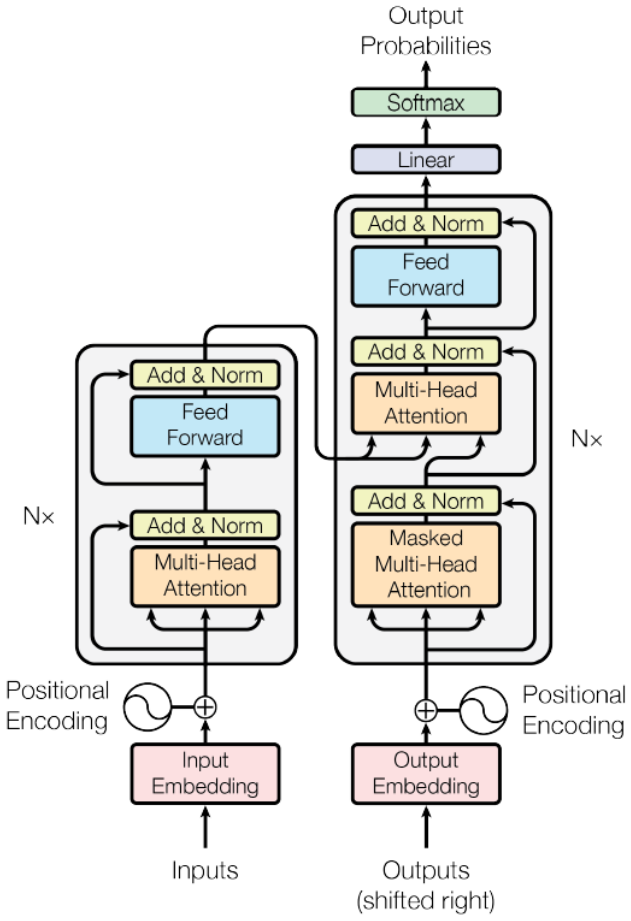


Figura 2.6. Architettura encoder-decoder di un modello Transformer [38].

Multi-Head Attention

L'attention è una misura per denotare la vicinanza semantica di una certa parola rispetto a un'altra. Nel contesto dei Transformer è detta *self-attention* perché, per ogni token, essa viene calcolata sugli altri elementi dello stesso testo. Ogni embedding viene proiettato (con trasformazioni lineari) in 3 vettori diversi, detti Q (query), K (key) e V (value). Per

ogni coppia Q, V, viene calcolata l'attenzione attraverso un prodotto scalare normalizzato dalla radice della lunghezza dei vettori, motivo per il quale essa è detta *Scaled Dot-Product Attention*. Il valore ottenuto viene infine normalizzato attraverso la funzione softmax, la formula dell'attention è quindi:

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

dove d_k è la lunghezza dei vettori Q e K.

Calcolando l'attention per ogni coppia Q, K si ottiene una matrice W, che viene usata come matrice di pesi sui vettori V per produrre i nuovo embedding. Se si indica con $v_1, ..., v_n$ il vettore dei values V, il nuovo embedding i-esimo viene quindi calcolato nel seguente modo:

$$x'_i = \sum_j w_{ji} v_j.$$

Le operazioni descritte coinvolgono matrici, per questo motivo il calcolo dell'attention può essere ottimizzato attraverso algoritmi efficienti di calcolo matriciale.

L'attention non viene in realtà calcolata come una singola funzione, ma il processo precedentemente illustrato segue un meccanismo detto *Multi-Head Attention*. Esso viene cioè effettuato h volte, ognuna con una trasformazione lineare indipendente dalle altre, producendo h attention diverse. I parametri delle trasformazioni lineari non vengono decisi a priori ma sono anch'essi parametri di training. Per avere un unico output, le diverse attention devono poi essere concatenate e nuovamente proiettate.

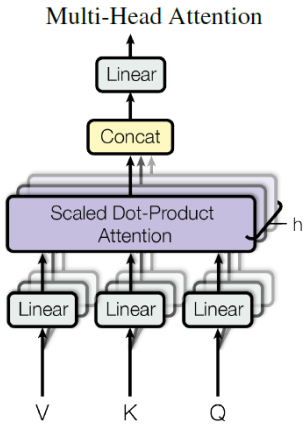


Figura 2.7. La Multi-Head Attention consiste in diversi livelli di attention che vengono calcolati parallelamente [38].

Nell'architettura Transformer con encoder-decoder, il meccanismo dell'attention viene utilizzato in 3 fasi:

- Come prima componente dell'encoder, in cui Q, K e V provengono tutti dalla stessa origine che è l'embedding dell'input iniziale.

- Come prima componente del decoder, in cui, similmente all'encoder, Q, K e V derivano dall'embedding dell'output, ma con l'accortezza di mascherare le connessioni verso i token che non sono ancora stati prodotti come output dal modello, per evitare che l'informazione sull'output da produrre si propaghi ovunque.

- Come componente che collega l'output dell'encoder al decoder. In questo caso, le query Q provengono dal livello precedente del decoder, mentre K e V provengono dall'output dell'encoder.

Reti Feed-Forward

La seconda componente dell'architettura Transformer è costituita da una semplice rete neurale Feed-Forward a due livelli. Nel paper originale la funzione di attivazione è una ReLU, ma più in generale è comune utilizzare funzioni GELU [13]. La particolarità di questa rete è che non viene applicata al vettore di token, ma sui singoli token separatamente, per questo motivo è detta *position-wise*.

Positional Encoding

Dal momento che la rete neurale costruita non presenta ricorrenze o convoluzioni, il modello non è in grado di utilizzare l'informazione sull'ordine delle parole. A questo proposito è quindi necessario aggiungere agli embedding l'informazione sulla posizione del token in considerazione. Questo viene fatto sommando all'embedding un vettore della stessa lunghezza che rappresenta la posizione nel testo del token considerato. In questo modo gli embedding somma incapsulano anche l'informazione sull'ordine delle parole. Possono essere fatte diverse scelte su come rappresentare la posizione nel vettore da sommare, nel paper originale sono utilizzate funzioni seno e coseno [38].

2.2.3 Transfer Learning

Il training di una rete neurale con architettura Transformer è un'operazione computazionalmente molto pesante. Esiste però una metodologia di training detta *Transfer Learning* che consente di diminuire lo sforzo computazionale necessario ad adattare i modelli ai propri dati. La procedura consiste nell'effettuare un primo training su una vastissima quantità di dati, con lo scopo di addestrare le abilità generali del modello. Queste step è detto *Pre-Training* e può essere eseguito dalle stesse società che creano il modello, di modo da distribuirlo con i parametri già allenati. Il passaggio successivo consiste nel applicare il modello pre-addestrato a dati più specifici, effettuando un nuovo training. Questa seconda operazione è detta può essere svolta in 2 modalità diverse: la *Feature extraction* e il *Fine tuning*.

La Feature extraction consiste nel suddividere i parametri del modello in due gruppi, il *corpo* e la *testa*. I parametri del corpo vengono allenati in fase di Pre-Training. In fase di training su dominio specifico, i pesi del corpo vengono congelati e si allenano solamente quelli della testa. Questo approccio consente al modello di essere flessibile e di poter essere allenato velocemente su domini specifici.

Con il Fine tuning, invece, i pesi della testa non vengono bloccati, ma tutti i parametri del modello possono essere ottimizzati. Il modello inizia il proprio training con i pesi di partenza derivanti dal Pre-Training e li aggiorna adattandoli al dominio specifico. Questa operazione è computazionalmente più pesante rispetto alla feature extraction, in quanto tutti i pesi del modello devono essere aggiornati, ma produce tendenzialmente risultati migliori.

Il Pre-Training è quindi la fase in cui il modello viene allenato per apprendere le caratteristiche generali di un linguaggio. In questo modo, nel dominio specifico, il training sarà, sia nel caso di Feature Extraction che nel caso di Fine Tuning, molto più agevole, in quando il modello dovrà apprendere solamente i pattern sui dati più specifici, senza dover ottimizzare i parametri in base alle regole generali di un linguaggio.

2.3 Text classification

La classificazione testuale è l'insieme degli algoritmi di apprendimento supervisionato che hanno lo scopo di assegnare a un testo di input la propria classe di appartenenza.

2.3.1 Feature extraction

Questo approccio consiste nel considerare gli embedding relativi a una collezione di documenti testuali e utilizzarli all'interno di algoritmi di classificazione. Questa è la metodologie che viene applicata con gli embedding generati dall'architettura Doc2vec. Essi possono infatti essere dati in input ad algoritmi di apprendimento statistico, come Random Forest o SVM, oppure essere utilizzati come input di reti neurali per la classificazione.

La Feature extraction è una metodologia che, come anticipato, può essere effettuata anche con architetture Transformer. La componente necessaria è quella dell'encoder, che ha come output il layer nascosto, contenente le rappresentazioni vettoriali di tutti i token di input. Ciò che viene fatto nei modelli di classificazione basati su Transformer (come BERT [6] e i suoi derivati) è di considerare, in particolare, l'embedding del token speciale [CLS] posto all'inizio di ogni sequenza. Esso infatti cattura il significato complessivo della frase e può essere utilizzato come input di un algoritmo di classificazione. Anche in questo caso, può essere utilizzato qualsiasi algoritmo di Apprendimento Statistico o di Deep Learning, detto *classification head*. La funzione di loss tipicamente utilizzata è la cross-entropy loss per la classificazione multi-classe (Appendice A).

Uno dei vantaggi dell'approccio basato su Feature extraction è la velocità di training, dato che il modello utilizzato per costruire gli embedding, che costituisce la parte più complessa e computazionalmente pesante, non deve essere considerato per la classificazione. Dopo che esso ha prodotto gli embedding necessari, gli unici parametri che dovranno essere allenati sono quelli della classification head. Il limite di questo procedimento risiede però nella separazione dei due task; gli embedding vengono prodotti indipendentemente dalle categorie su cui si deve eseguire la classificazione e questo può essere causa di risultati meno accurati rispetto a quelli forniti dal Fine tuning.

2.3.2 Fine tuning

L'architettura Transformer consente di essere allenata su task specifici, come quelli di classificazione, attraverso il Fine tuning. Questa metodologia consiste nell'integrare la rappresentazione degli embedding e la classificazione all'interno di un unico modello, per il quale tutti i parametri sono liberi di essere ottimizzati durante l'addestramento del classificatore, Questo approccio richiede di eseguire una discesa del gradiente su tutti i parametri, per questo motivo, la classification head deve essere differenziabile ed è perciò costituita da una rete neurale. Anche in questo caso viene come input della classification head l'embedding del token speciale [CLS]. Essa, così come nel caso di Feature extraction, trasforma il vettore n-dimensionale [CLS] (dove n è la dimensione dello stato nascosto) in una distribuzione di probabilità sulle c classi di output. Lo svantaggio di questo approccio è di essere computazionalmente pesante, dal momento che tutti i parametri devono essere ottimizzati. Il vantaggio è di integrare due task in un unico modello, producendo generalmente risultati più accurati.

In generale, la classificazione di input testuali richiede solamente di avere una rappresentazione vettoriale dei testi. La componente di decoding dell'architettura Transformer non è perciò utile in questa tipologia di task, mentre svolge un ruolo fondamentale in tutte le attività che richiedono la generazione di testo, come verrà illustrato nella sezione successiva.

2.4 Text summarization

Il riassunto automatico di testi rientra nella più ampia categoria del NLP detta *Text generation*. In questa sezione, si illustra in che modo si può produrre testo utilizzando l'architettura Transformer. Fino a questo punto si è spiegato in che modo un Transformer può produrre embedding e generare una distribuzione di probabilità sui token di output. In particolare, il training del Transformer nel contesto delle generazione testuale avviene mediante gli output del decoder. Sia i coefficienti del decoder sia quelli dell'encoder vengono ottimizzati come conseguenza della backpropagation. Nel processo di training, i modelli utilizzano una forma di cross-entropy loss applicata sulla distribuzione di probabilità delle parole di output. Essa è descritta da:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{L_i} y_{ij} \cdot \log(p_{ij}),$$

dove:

- N è il numero di sequenze di addestramento
- L_i è la lunghezza della i-esima sequenza
- y_{ij} è il vettore one-hot encoding per il token del testo i in posizione j

- p_{ij} è la distribuzione delle probabilità di output sui token del dizionario per il j-esimo token del testo i.

L'obiettivo della text generation è utilizzare gli output probabilistici per selezionare le parole per la produzione di un testo. L'architettura encoder-decoder consente di individuare la parola più probabile a un certo punto della sequenza. In fase di applicazione del modello, l'output prodotto non è però costituito dalla semplice sequenza di parole più probabili, ma viene utilizzata una strategia più sofisticata, descritta nella sezione seguente.

2.4.1 Beam search

Dato un certo input, l'obiettivo di un sistema di text generation è di produrre la sequenza di parole con la massima probabilità. In altri termini, data la sequenza di contesto x, si vuole cercare la sequenza di output y tale che

$$y = \text{argmax}[P(y|x)],$$

sulla base delle probabilità predette dal modello. Se il vettore di output y è costituito da N token, la probabilità di y condizionata a x si può decomporre usando la regola della catena per probabilità condizionate:

$$P(y|x) = \prod_{t=1}^{N} P(y_t|y_1, ..., y_{t-1}, x). \tag{2.1}$$

Trovare l'ottimo di questa probabilità è però un problema computazionalmente troppo pesante, in quanto ha complessità esponenziale. Si ricorre perciò a un euristica che consente di trovare un approssimazione della soluzione cercata, ovvero la *Beam Search*.

La Beam Search è una metodologia consolidata nel campo NLP, il cui nome è stato coniato nel 1977 dal dipartimento di informatica della Carnegie-Mellon University [37], e che ha sempre avuto largo impiego nei sistemi decoding. Se ne descrive ora il funzionamento generale, seguendo [36]. Si considera, anzitutto, la decomposizione (2.1). La Beam Search consiste nell'ottimizzare tale probabilità scegliendo un numero b di *beam* e considerando solamente i primi b token più probabili, per ogni step. Questo significa che per ottimizzare la probabilità bisognerà cercare tra b^N possibili sequenze, invece che tra d^N, dove d è la dimensione del dizionario di token, rendendo quindi il calcolo molto più agevole. In particolare, per scegliere la sequenza di token con la probabilità maggiore, si mettono a confronto le log-probabilità, invece delle probabilità semplici. Il motivo è dovuto a ragioni computazionali, in quanto successive moltiplicazioni tra numeri più piccoli di 1 possono portare a risultati inferiori alla sensibilità del calcolatore, causando underflow aritmetico. Utilizzando invece il logaritmo delle probabilità, che è una funzione monotona, il prodotto diventa una somma:

$$\log[P(y|x)] = \sum_{t=1}^{N} \log[P(y_t|y_1, ..., y_{t-1}, x)],$$

consentendo di selezionare la sequenza con la massima log-probabilità senza causare underflow.

Si fa notare come, dato che le log-probabilità sono una somma di valori negative, le sequenze di lunghezza maggiore hanno tendenzialmente una valori inferiori rispetto a quelle più corte. Per questo motivo si può dividere il valore della log-probabilità per la lunghezza della sequenza, in modo da non poter confrontare sequenze con lunghezze diverse.

Nei sistemi di text summarization si è interessati a non avere output troppo lunghi. A questo proposito viene indicata una lunghezza massima dell'output e si utilizza un coefficiente α (detto *length penalty*) per gestire la scelta tra sequenza con lunghezze diverse. Questo coefficiente viene utilizzato come esponente della lunghezza della sequenza posta a denominatore:

$$\frac{\log[P(y|x)]}{N^{\alpha}}$$

Ponendo $\alpha < 1$ si favorisce perciò la scelta di sequenze con lunghezza minore.

Capitolo 3

Dataset

Questo capitolo ha lo scopo di illustrare nel dettaglio il dataset utilizzato per gli esperimenti. Se ne descriveranno anzitutto le caratteristiche generali e la struttura, si fornirà poi un'analisi esplorativa e saranno infine presentate le operazioni di preprocessing effettuate su di esso.

Si specificano in primo luogo alcune operazioni eseguite sui dati direttamente provenienti dal database FIDA per estrarre il dataset utilizzato. Per indicare più specificamente l'effettiva istanza di un fondo d'investimento si può utilizzare il termine *comparto*. In origine sono presenti i dati di 66953 comparti. Questo dataset presenta però pesanti ridondanze per il lavoro da sviluppare, in quanto lo stesso comparto può essere distribuito in diversi modi, con diverse valute o emittenti e ogni riga del dataset rappresenta una di queste diverse distribuzioni. Uno degli attributi presenti nel dataset (descritti nella sezione 3.2) è la valuta. Per preservare la corretta informazione, è importante che comparti con valute diverse siano item del dataset distinti. Per questo motivo, la primissima operazione viene eseguita da una funzione che estrae un sotto dataset che per ogni coppia comparto/valuta contenga un solo item. In questo modo si ottiene un dataset molto più piccolo, che riduce le ridondanze, con 18998 fondi.

Ai fini del lavoro, i fondi privi di campi testuali sono inutili e per questo sono stati eliminati. Inoltre, è necessario definire una lingua con cui lavorare ed eseguire tutti gli algoritmi di NLP. Si è scelta la lingua italiana, dal momento che il 93.65% dei fondi con campi testuali non nulli è in italiano, ottenendo un dataset di 12914 righe.

Come riportato al Capitolo 1, nella descrizione del sistema di classificazione, tra gli attributi relativi al fondo vi è l'asset, cioè la tipologia di strumenti in cui il fondo può investire. La metodologia sviluppata nella tesi per migliorare le performance del classificatore è stata definita nel caso delle due asset class più ampie: Azionari e Obbligazionari. Per questo motivo, ai fini del lavoro, si considerano solamente i dati appartenenti a queste due macro, costituiti da 8855 fondi. È quindi evidente come siano state fatte scelte abbastanza selettive in termini di dati da trattare. Le ragioni sono principalmente legate alla praticità di sviluppo della metodologia presentata nel Capitolo 4. Esse sono inoltre assunzioni che non perdono di generalità, in quando il lavoro può essere esteso, con le dovute precauzioni, anche alle altre asset class e ai testi in altre lingue. Inoltre, il lavoro svolto non ha finalità operative, ma ha scopo di ricerca, per comprendere le potenzialità e

i limiti offerti dal NLP. Perciò, la selezione del preciso dataset di lavoro consente di avere un ambiente controllato in cui valutare la bontà degli esperimenti effettuati.

Vi sono due ultime selezioni eseguite, di carattere pratico. La prima riguarda la selezione dei soli fondi la cui categoria di appartenenza contenga almeno 6 elementi. Questa scelta è necessaria per poter suddividere i dati in train, validation e test set ai fini della classificazione. La seconda riguarda l'eliminazione dei fondi appartenenti a una delle cosiddette *categorie hedged*, che sono fondi il cui rischio d'investimento viene mitigato attraverso una particolare esposizione valutaria. Questi fondi infatti, sono, a livello testuale, qualitativamente identici ai rispettivi fondi della stessa categoria non *hedged*, per cui non ha senso considerare categorie diverse per lo stesso dato. In conclusione, il dataset utilizzato per tutti gli esperimenti contiene 7373 fondi.

3.1 Panoramica sui dati

Gli elementi del dataset hanno 32 attributi descrittivi. Essi possono essere ricondotti a una delle seguenti tipologie:

- attributi anagrafici
- attributi relativi alla politica di investimento
- campi della tabella di classificazione
- categorie attribuite dalla categorizzazione.

Alcuni attributi (quasi tutti quelli anagrafici) non sono utili ai fini della tesi. Quelli considerati sono 20 e sono i seguenti:

[*ana_name, pol_testo, pol_finalita, asset, geo_area, geo_zone_country, sector,sub_sector, dimension, style, maturity, rating, type, issuer, currency_exposure, asset_exposure, categoria, microclasse, classe, macro*]

Gli attributi relativi alla politica di investimento sono costituiti dai campi testuali utilizzati nel lavoro, a cui si aggiunge il nome del fondo, che è un dato anagrafico (l'unico utilizzato).

- *pol_testo* riassume la politica di investimento, cioè in che tipologie di asset il fondo può investire e con quali modalità.
- *pol_finalita* riassume gli obiettivi del fondo, specificando quali sono le performance che il fondo cerca di realizzare.
- *ana_name* è il nome del fondo.

I campi della tabella di classificazione costituiscono la maggioranza degli attributi e sono tutti i campi presenti nella tabella di classificazione 1.1.

Vi sono infine le categorie relative al sistema di categorizzazione, cioè *Macro*, *Classe* e *Categoria* secondo i diversi livelli di dettaglio. Anche questo tipo di attributo è stato già descritto al Capitolo 1, nella sezione 1.2.2.

3.2 Analisi esplorativa

Si illustrano ora alcune caratteristiche quantitative dei testi di input e delle categorie di output. Per quanto riguarda i testi di input, si mostrano alcune statistiche sulla loro lunghezza. Nella tabella seguente si riportano media e deviazione standard sulla lunghezza dei vari attributi espressa sia in caratteri che in parole.

	Parole		Caratteri	
	Media	Std	Media	Std
ana_name	7.94	1.94	43.87	10.04
pol_testo	105.25	38.42	723.40	264.26
pol_finalita	27.66	16.60	188.89	113.86

Il campo testuale più corposo è rappresentato da *pol_testo*. L'attributo *ana_name* è invece il più corto, esso contiene infatti solo il nome del fondo, che è un insieme di parole che lo descrive molto brevemente.

Nonostante si stia già considerando il dataset ripulito dai dati mancanti, si riporta che, in origine, il campo relativo al nome del fondo è presente in tutti gli item. Questo è coerente con il fatto che, quando un fondo viene inserito nel database FIDA, *ana_name* viene sempre inserito, in quanto definisce il fondo stesso, mentre gli altri campi testuali non sono sempre presenti, in quanto devono essere estratti dalla documentazione. In particolare il campo *pol_finalita* ha una percentuale di dati mancanti del 23.60% e il campo , molto simili a quella di *pol_testo* con il 23.62%.

Si riportano nel seguito gli istogrammi che mostrano le distribuzioni delle lunghezze dei testi.

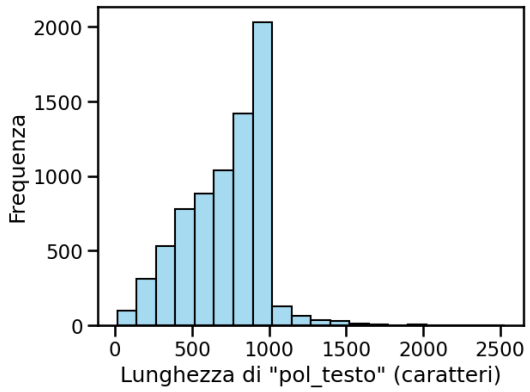


Figura 3.1. Istogramma sulla lunghezza in caratteri di *pol_testo*.

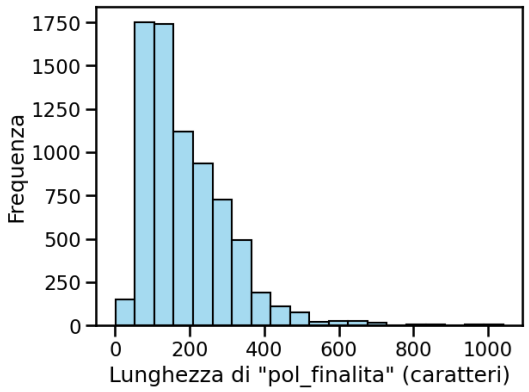


Figura 3.2. Istogramma sulla lunghezza in caratteri di *pol_finalita*.

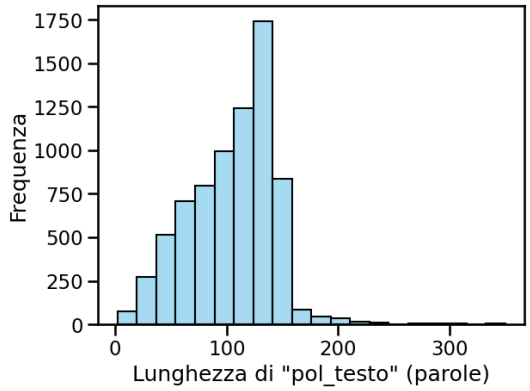


Figura 3.3. Istogramma sulla lunghezza in parole di *pol_testo*.

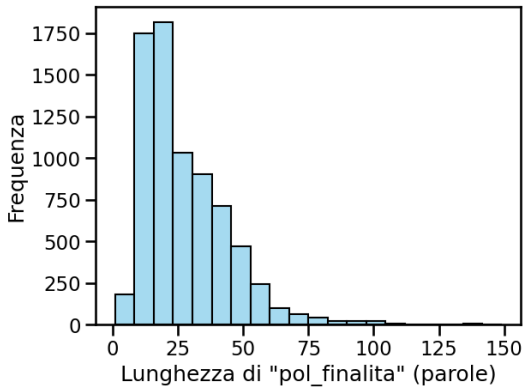


Figura 3.4. Istogramma sulla lunghezza in parole di *pol_finalita*.

Si descrivono ora alcune caratteristiche sulle categorie di output. Nonostante il processo di categorizzazione consista in una classificazione gerarchica su tre livelli di dettaglio, il livello di categorizzazione di maggiore interesse è costituito dalle *categorie*, il livello di dettaglio maggiore. Infatti, tutte le considerazioni e le analisi che possono poi essere eseguite sui fondi, vengono effettuate sulla base della loro categoria di appartenenza. Per il dataset considerato, sono presenti 160 categorie diverse che, su un totale di 7373 fondi, con una media di 46.08 fondi per categoria. La distribuzione del numero di elementi sulle categorie è però molto irregolare, come si può vedere nel seguente istogramma.

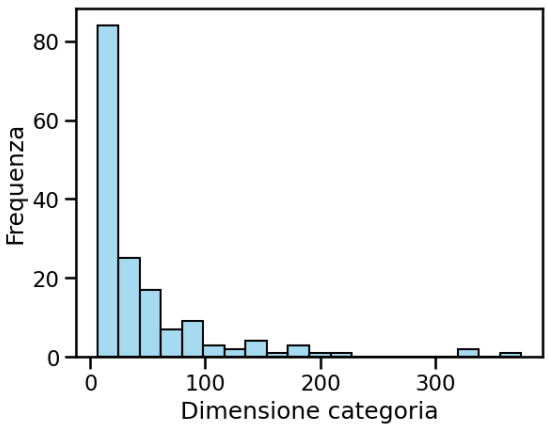


Figura 3.5. Istogramma della dimensione delle categorie sulle categorie originali.

Si può notare infatti come vi sia una netta maggioranza di categorie con pochi elementi. Vi sono infatti solamente 18 categorie che contengono un numero di elementi superiore a 100 (11.25%), mentre quelle con meno di 25 elementi sono 84 (52.5%).

3.3 Preprocessing

Sono state effettuati alcuni step di preprocessing sul dataset utilizzato. Le prime operazioni eseguite sono la normalizzazione di testi e attributi e la gestione di particolari categorie dette *di chiusura*. Queste operazioni sono state effettuate su tutti i dati, a prescindere dal modello utilizzato. Il processo di tokenizzazione dipende invece dal modello utilizzato. Con Doc2vec vengono selezionate e preprocessate le singole parole, mentre con le architetture Transformer si possono usare diverse tecniche per individuare i token come parole o sotto-parole.

3.3.1 Normalizzazione testi e attributi

Pulizia testi

Come prima operazione, viene effettuato una pulizia dei testi, attraverso una funzione che li normalizza. Infatti, nei testi finanziari sono spesso presenti sigle o simboli, che è preferibile avere sotto forma di parole. Ci si riferisce in particolare a simboli e sigle di monete come € e $ per euro e dollaro o CHF per il franco svizzero. Un altro tipo di normalizzazione riguarda sigle che si riferiscono a parole specifiche. In particolare è comune utilizzare le sigle *H* o *Hdg* per riferirsi al termine *Hedged* e la sigla *Eq* per riferirsi a *Equity*.

Il lavoro sviluppato nel seguito consiste in un sistema di classificazione testuale a partire dagli attributi testuali *pol_testo* e *pol_finalita*. Questi attributi contengono informazioni diverse tra loro, ma ugualmente importanti ai fini della classificazione e sono sempre stati dati in input contemporaneamente. Per questo motivo, si è deciso di accorparli in un unico attributo testuale detto *testo_finalita*, attraverso una concatenazione testuale.

Normalizzazione attributi

Un importante step di preprocessing riguarda la normalizzazione degli attributi relativi alla tabella di classificazione. Questi attributi saranno infatti utili per la strategia di miglioramento del sistema di classificazione illustrata nel Capitolo 4. È perciò importante che essi abbiano un formato adeguato per le operazioni successive. In particolare, vi sono valori assunti da alcuni attributi che, in contesti specifici, non portano nessuna informazione rilevante e che sono stati quindi impostati sul valore *None*. Si illustrano, nel seguente elenco, tutte le situazioni a cui ci si riferisce.

- *issuer* con asset *Azionari*: gli emittenti delle azioni sono sempre aziende, per cui in questo caso l'attributo *issuer* assume sempre il valore *Corporate* e non è quindi informativo.

- *dimension* con asset *Obbligazionari*: al netto di errori di inserimento, l'attributo assume sempre i valori *Non significativo* o *Non applicabile*.

- *maturity* con asset *Azionari*: per loro natura gli asset azionari non hanno scadenze, il campo *maturity* è sempre *non applicabile*

- *rating* con asset *Azionari*: il rating riguarda la capacità di solvibilità di debiti da parte dell'emittente, non è un fattore significativo per asset azionari.

- *style* con asset *Obbligazionari*: le obbligazioni non hanno stili particolari, al netto di errori il campo è sempre su *Non significativo* o *Non applicabile*.

- Il valore degli attributi *geo_zone_country* e *geo_area* viene gestito in modo da avere informazioni utili senza eccessiva ridondanza. Quando i valori assunti da *geo_zone_country* appartengono a un insieme di zone sufficientemente specifiche, allora si è scelto di non considerare il valore di *geo_area* in quanto poco informativo. Negli altri casi, sono stati tenuti entrambi i campi.

- *sub_sector* con asset *Obbligazionari*: Si è deciso di escluderlo in quanto è quasi sempre poco significativo. Per quanto riguarda invece il sector (con asset Obbligazionari) si è decido di escluderlo solamente in presenza di campi non esplicativi (*Non Significativo, Non Applicabile, Settori Multipli*).

- *sector* e *sub_sector* con asset *Azionari* sono stati gestiti in modo particolare. Il campo *sub_sector* non è stato considerato nel caso in cui assume il valore *Sottosettori Multipli*. Il campo *sector*, invece, è stato eliminato ogni volta che assume il valore *Settori Multipli*, ma il *sub_sector* è diverso da *ESG*.

Ponendo il valore *None* in tutti casi citati si ottiene un dataset con attributi meglio trattabili dai modelli utilizzati nel seguito. La scelta di lavorare con i soli dati che hanno come asset class *Azionari* o *Obbligazionari* deriva dal riconoscimento dei pattern appena descritti. Per estendere il lavoro anche ad altri tipi di asset class, è necessario riconoscere le relazioni per normalizzare gli attributi di modo da avere una base dati opportunamente informativa.

3.3.2 Categorie di chiusura

Nel database FIDA sono presenti alcune categorie, dette *categorie di chiusura* che contengono fondi non ascrivibili a categorie con caratteristiche omogenee tra loro, ma che hanno la funzione di unire fondi specifici in insiemi che li contengano. Anche nel sotto dataset utilizzato per tutti gli esperimenti sono presenti alcune di queste categorie, come ad esempio le categorie *Altri Azionari* o *Altri Obbligazionari*. Per lo sviluppo di un sistema di classificazione, non ha senso considerarle al pari delle altre, dal momento che i fondi presenti al loro interno non condividono caratteristiche sufficientemente omogenee tra loro. Per questo motivo, queste categorie sono state unite in un'unica categoria speciale chiamata *non categorizzato*. Queste categorie, individuate in azienda, sono 18 per quanto riguarda il dataset utilizzato. Accorpandole in un unica categoria, il numero totale di categorie passa perciò da 160 a 143. La nuova categoria costruita è molto consistente, contiene infatti 2790 elementi, cioè il 37.8% degli elementi totali. Sia per la sua preponderanza sul totale e sia per la sua natura, tale categoria sarà considerata in modo particolare nella definizione delle metriche di valutazione del classificatore.

Nel seguito si denoterà la categoria *non classificati* con il termine *categoria di chiusura*, mentre con *categorie vere* ci si riferirà alle altre categorie. Si mostra ora un istogramma che illustra la distribuzione della dimensione delle categorie vere.

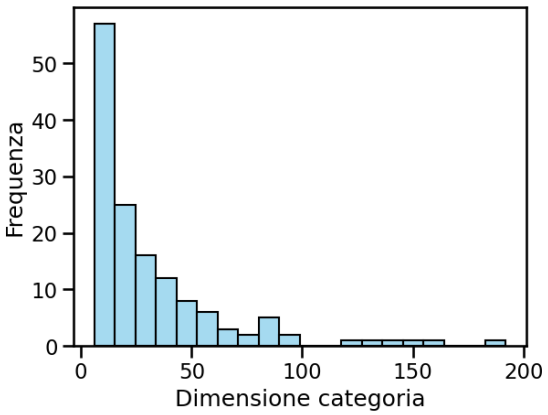


Figura 3.6. Istogramma della dimensione delle categorie esclusa quella di chiusura.

Anche ad esclusione della categoria di chiusura, il dataset continua ad essere sbilanciato, seppur in maniera leggermente minore rispetto alle categorie originali in Figura 3.5.

Infine, il dataset è stato suddiviso in test, validation e train set, gli stessi per tutti gli esperimenti effettuati. Questa suddivisione è stata eseguita attraverso uno stratified sampling rispetto all'attributo *categoria*, con proporzioni 70:15:15 per test, validation e train rispettivamente.

3.3.3 Tokenizzazione per Word2vec

Il processo di creazione di token è lo step che consente di avere dati processabili da un determinato modello. Per questo motivo esso può essere diverso a seconda del modello da utilizzare. Nel caso del modello Word2vec, i token sono costituiti da intere parole. Prima di scomporre il testo in token vengono però applicate ulteriori operazioni di pulizia testo, come l'eliminazione della punteggiatura e la rimozione delle stopword della lingua italiana.

Dopo il processo di pulizia sono state applicate le seguenti operazioni per la creazione dei token.

- Ogni stringa viene trasformata in una lista costituita dalle parole che la compongono.
- Come descritto alla sezione 2.1.1, per ridurre la grande variabilità di parole, esse vengono ricondotte alla propria radice attraverso uno stemmer basato su lemmatizer (entrambi costruiti specificamente per la lingua italiana).

- Infine, le parole con lunghezza minore o uguale a 2 e maggiori o uguali a 16 vengono eliminate, in quanto è molto probabile che si tratti di errori. Infatti, le prime appartengono alla lista di stopword e, se ve ne sono altre presenti, allora non sono di senso compiuto. Le seconde, invece, se presenti, derivano probabilmente dalla mancata presenza di spazi tra due parole.

Grazie a queste procedure si ottiene il dizionario di token da utilizzare, ogni token del dizionario viene trasformato in un vettore di input con il one-hot-encoding. Gli svantaggi del processo descritto consistono sostanzialmente nella perdita di informazioni sul testo a causa delle operazioni di pulizia subite, che sono però necessarie per avere una dimensione dell'input non eccessivamente elevata.

Le architetture Transformer consentono di catturare più relazioni logiche e semantiche tra gli elementi del testo e la procedura di tokenizzazione avviene in maniera diversa.

3.3.4 Tokenizzazione per Transformer

Come anticipato alla sezione 2.2.1, con i modelli Transformer, i testi di input non richiedono particolari operazioni di pulizia. La varietà linguistica viene gestita attraverso sistemi di tokenizzazione basati sulle sotto parole (subword). Si illustrano le tre tipologie di subword tokenization impiegate dai modelli Transformer utilizzati nel lavoro.

WordPiece

La tokenizzazione WordPiece è stata introdotta per la prima volta nel 2012 in [31] per la ricerca vocale in Giapponese e Coreano. Nel contesto dei Transformer si fa riferimento all'adattamento presentato in [42]. Vi è inizialmente una fase di training del modello WordPiece, che consiste nella creazione di un dizionario di sotto parole a partire dai singoli caratteri che vengono iterativamente uniti in modo da costruire le sotto parole più frequenti. Una volta costruito questo dizionario, può essere applicata la tokenizzazione delle parole del testo di interesse, attraverso una logica longest-match-first [7]. Questo avviene nel seguente modo, per ogni parola:

- Si seleziona, a partire dall'inizio della stringa, la sotto parola più lunga contenuta nel dizionario.

- Se al punto precedente si trova una sotto parola valida (contenuta nel dizionario) la si aggiunge all'elenco dei token. Si considera poi la porzione di stringa rimanente e si itera il punto precedente.

- Se non viene trovata nessuna parola valida, la stringa viene inserita nella lista dei token sconosciuti.

Questa metodologia consente di avere un numero di token che è al più pari alla dimensione del dizionario di partenza. Le porzioni di parole che non vengono riconosciute dal dizionario vengono però scartate.

Byte Pair Encoding

Il Byte Pair Encoding presentato originariamente in [10] e adattato per la segmentazione di parole in [33] consiste nel costruire il dizionario di token attraverso i seguenti passaggi:

- Viene inizializzato un dizionario di simboli costituito da tutti i caratteri presenti nel testo.
- Si seleziona la coppia di caratteri con maggiore frequenza nel testo e si crea una si un nuovo simbolo dato dalla coppia.
- Si itera il punto precedente fino a quando il numero di simboli totale non raggiunge la dimensione desiderata.

L'approccio è quindi simile al WordPiece, ma a differenza di quest'ultimo il dizionario di token viene costruito direttamente a partire dai caratteri del testo. Al contrario, in WordPiece si procede inizialmente con la fase di training per costruire un dizionario di token e le parole del testo desiderato vengono successivamente scomposte a partire dai token del dizionario.

Unigram

La tecnica di tokenizzazione Unigram (anche detta SentencePiece), è una metodologia presentata nel 2018 in [16]. La procedura comincia con l'inizializzazione di un dizionario di sotto parole. Esso può essere costruito a partire dall'insieme di tutti i caratteri più le sotto parole più frequenti nel corpus testuale di training (operazione che può essere eseguita attraverso il bpe). Successivamente, per applicare la tokenizzazione al testo di interesse viene iterata la seguente procedura fino a quando il dizionario di token $\mathcal{V}$ non raggiunge la dimensione desiderata:

- Viene massimizzata la seguente funzione:

$$x^* = \arg \max_{x \in S(X)} \prod_{i=1}^{M} p(x_i),$$

  dove le x_i sono le sotto parole del testo presenti nel dizionario (di dimensione M) in corso e p è la distribuzione di probabilità empirica. Questo problema di ottimizzazione viene risolto con l'algoritmo di Viterbi [39].

- Viene calcolata la $loss_i$ per ciascuna sotto parola x_i, dove $loss_i$ indica di quanto la verosimiglianza $\mathcal{L} = \sum_{s=1}^{|D|} \log\big(P(X^{(s)})\big)$ (dove le s sono le sotto parole) si riduce se il token x_i viene rimosso dall'attuale vocabolario.
- I token vengono ordinati in modo decrescente sulla base di $loss_i$ e si seleziona il primo $\nu\%$. Questi token costituiscono il nuovo dizionario con cui si itera il processo.

In questo modo, a partire da un dizionario di partenza di dimensione elevata, si selezionano solamente le sottoparole più rilevanti nel testo di interesse.

Capitolo 4

Metodologia proposta

Dopo aver descritto il contesto e i dati con cui si opera e aver illustrato le basi teoriche delle tecnologie utilizzate, in questo capitolo si presentano gli obiettivi e la metodologie del lavoro svolto. Lo scopo del lavoro è di sviluppare un sistema di classificazione testuale per fondi d'investimento. Oltre all'implementazione degli algoritmi di classificazione sui dati aziendali, l'obiettivo del lavoro è migliorare i risultati ottenuti, attraverso la text summarization.

Si descriverà, in primo luogo, la metodologia per lo sviluppo di un sistema di text summarization supervisionato e in che modo esso è stato integrato nel sistema di classificazione. Verrano poi presentati i diversi modelli di text classification e text summarization che sono stati utilizzati.

4.1 Text classification basata su summarization

I dati testuali utilizzati consistono nell'attributo derivante dalla concatenazione degli attributi su politica d'investimento e finalità del fondo (*pol_testo_finalita*). Tale attributo è abbastanza lungo e dispersivo, ma contiene molte informazioni sul fondo in questione. L'obiettivo del lavoro è sviluppare un sistema di classificazione testuale che possa sfruttare al meglio le informazioni contenute in questo testo.

L'idea sviluppata consiste nel generare un riassunto automatico del testo, in modo da condensare le informazioni principali. Successivamente si possono applicare gli algoritmi di classificazione sui riassunti generati, con l'obiettivo di avere performance migliori rispetto a quelle ottenute con i testi originali. Si vuole cioè ridurre la lunghezza dei testi sui cui eseguire la classificazione, preservando le informazioni rilevanti, in modo da eliminare l'eccessivo rumore presente nei testi originali. Per poter generare riassunti in maniera automatica è stato implementato un sistema di text summarization, basato sulle architetture Transformer.

La generazione automatica di riassunti può inoltre essere utile anche al di fuori del sistema di classificazione dei fondi. Poter generare riassunti automatici di documenti può infatti essere un operazione utile ai fini aziendali, per velocizzare le procedure di analisi dei testi o per generare report.

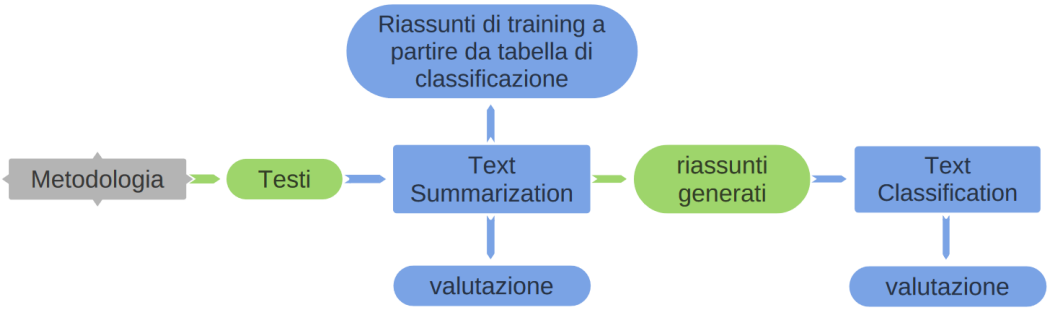


Figura 4.1. Rappresentazione degli step della metodologia utilizzata.

I modelli di text summarization utilizzati devono essere allenati attraverso un Fine tuning, dal momento che essi sono addestrati per eseguire riassunti generici e non sono in grado di individuare le informazioni utili ai fini della classificazione del fondo. È stato perciò necessario trovare una strategia per ottenere dei riassunti di riferimento su cui poter addestrare i modelli. Nella sezione successiva si illustra come sono stati costruiti.

4.1.1 Riassunti di riferimento

La mancanza di un insieme di riassunti di riferimento su cui effettuare il training, ha reso necessario lo sviluppo di un sistema che li costruisca sulla base delle informazioni presenti in database. In particolare, sono stati utilizzati i campi relativi agli attributi della tabella di classificazione 1.1 per costruire i riassunti desiderati. Tali informazioni sono infatti le caratteristiche principali sulla base delle quali sono definite le categorie. Questi attributi vengono estratti sulla base della documentazione (e quindi dei testi in database), per cui si vuole addestrare un algoritmo che sappia estrarre le informazioni principali condensandole in un riassunto, anche quando queste informazioni non sono ancora state estratte manualmente. Questo significa sviluppare un algoritmo di classificazione che richieda solamente la presenza dei dati testuali, estratti direttamente dalla documentazione, senza particolari operazioni.

La normalizzazione degli attributi eseguita in fase di preprocessing consente di poterli utilizzare nella costruzione dei riassunti di riferimento. Tali riassunti vengono costruiti concatenando gli attributi della classificazione attraverso diverse combinazioni di frasi in modo da generare un testo di senso compiuto. Ad esempio, l'inizio del riassunto di riferimento può iniziare con una delle seguenti frasi:

['Il comparto investe in', 'Il fondo investe in', 'Il comparto investe prevalentemente in', 'Il fondo investe principalmente in', 'Il comparto deterrà prevalentemente'],

la cui parola successiva viene estratta dal valore dell'attributo *Asset*. Sono state cioè generate delle possibili scelte di frasi per costruire un testo che presenti un certa varietà linguistica. La frase utilizzata viene selezionata sulla base di un campionamento uniforme tra quelle disponibili. La presenza del valore *None* nei casi in cui un attributo non

è informativo consente di individuare priori se esso va incluso all'interno del riassunto. Questa procedura di concatenazione tra frasi e attributi viene effettuata su tutti i campi della tabella classificazione, ad esclusione di:

- *currency_exposure*, in quanto essa è un'informazione che non è tipicamente presente nei testi da riassumere;
- *asset_exposure*, dal momento che nella porzione di dataset considerata, esso coincide con il campo asset (*Azionari* o *Obbligazionari*);
- *type*, dato che questa informazione è deducibile dal testo, ma non è utile ai fini del processo di classificazione, per cui si è scelto di non considerarla.

La seguente tabella mostra un esempio di possibili valori assunti dagli attributi.

Attributo	*Valore*
asset	Obbligazionari
geo_area	Globale / Aree multiple
geo_zone_country	Mercati sviluppati
sector	None
sub_sector	None
dimension	None
style	None
maturity	Breve Termine
rating	Classe di Rating Investment Grade
issuer	Government

Un possibile riassunto di riferimento generato con il sistema descritto è il seguente.

Il comparto investe prevalentemente in obbligazioni di tipo Government in società con sede in tutto il mondo, specialmente in Mercati sviluppati. I titoli appartegono a Classe di Rating Investment Grade e hanno scadenza a Breve Termine.

I riassunti di riferimento vengono costruiti per ogni fondo del dataset e vengono aggiunti in una nuova colonna chiamata *summary*.

4.2 Modelli di classificazione

In questa sezione si illustrano i diversi modelli di classificazione che sono stati utilizzati nella tesi e i cui risultati sono stati messi a confronto.

4.2.1 Doc2vec e SVM

Uno degli approcci per la classificazione anticipato al Capitolo 2 è quello basato sull'applicazione di algoritmi di classificazione sugli embedding prodotti da un modello. In

particolare, è stato applicato il modello Doc2vec ai testi di input e i vettori generati sono stati sottoposti a una classificazione con una Support Vector Machine (SVM) lineare. La SVM è l'unico algoritmo non basato su reti neurali che è stato utilizzato nel lavoro, per la sua descrizione si fa riferimento a [12].

Si considera il caso in di SVM applicata alla classificazione. Nel caso binario il problema è descritto da una serie di punti $x_i \in \mathbb{R}^N$, con $i = 1, ..., p$, ognuno dei quali assume valore $y_i \in \{1, -1\}$. L'idea della SVM è di classificare i dati sulla base di un iperpiano separatore del tipo

$$x^T\beta + \beta_0 = 0.$$

L'obiettivo è trovare il miglior iperpiano separatore, cioè quello che massimizza il margine, vale a dire la distanza minima tra il piano e un punto. In generale, i punti non sono linearmente separabili ed è perciò necessario introdurre delle variabili di rilassamento, che consentano ad alcuni punti di violare il confine definito dall'iperpiano. Come illustrato in [12], il problema può essere riscritto nel seguento modo.

$$\begin{aligned} \min_{\beta,\beta_0,\xi_i \in \mathbb{R}^+} \quad & \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s. t.} \quad & y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \quad i = 1, \ldots, N \\ & \xi_i \geq 0, \quad i = 1, \ldots, N \end{aligned}$$

Le ξ_i sono le variabili di rilassamento o C è l'iperparametro del modello che indica il livello di rilassamento applicato. Il problema appena scritto è di programmazione quadratica e può essere risolto utilizzato i moltiplicatori di Lagrange.

La generalizzazione al caso di classificazione multiclasse può essere fatta costruendo i vari iperpiani di separazione sulla base di una logica one-vs-one oppure one-vs-all. La versione della SVM che non fa uso di kernel [3] è detta SVM lineare.

4.2.2 BERT

Il modello BERT (*Bidirectional Encoder Representations from Transformers*) presentato nel paper [6] è stato il primo modello Transformer con la sola componente di encoding. Esso nasce con l'obiettivo di migliorare le performance dei modelli con architetture unidirezionali in fase di pre-training. Il pre-training di BERT è stato eseguito su due task generici non supervisionati: il *masked-language-modeling (MLM)* e la *next sentence predicion (NSP)*. Gli autori hanno utilizzato l'architettura del'encoder come descritta nel paper originale sui Transformer [38], con i seguenti parametri:

- L = 12 layer (blocchi del Transformer);
- dimensione dei layer nascosti (embedding) pari a H = 768;
- numero di head dell'attention pari a A = 12.

Questi parametri sono riferiti alla versione base di BERT, che è quella che è stata utilizzata. Si mostra ora in che modo gli autori hanno eseguito il pre-training sui task generici.

Masked Language Model

Questo tipo di addestramento consiste nel far prevedere al modello una certa porzione di token del testo dopo averli mascherati. In particolare, questo avviene sostituendo il 15% dei token (estratti casualmente) con il token speciale [MASK] e addestrando il modello a prevederli. Tuttavia, per evitare di avere un'eccessiva perdita di informazioni sui token mascherati e per creare robustezza nel modello, i token da mascherare non vengono sostituiti da [MASK] in tutti i casi. Il 10% viene sostituito da un token casuale, un altro 10% viene lasciato uguale e il restante 80% viene invece sostituito dal token speciale [MASK].

Next Sentence Prediction

Per questo task, vengono scelte coppie di frasi A e B, per le quali il 50% delle volte la frase B è la frase successiva ad A e per il 50% è una frase estratta casualmente. Nel primo caso la frase B viene denotata con *IsNext* e la seconda con *NotNext*. Il modello viene poi addestrato a prevedere quando la frase B è quella successiva e quando non lo è, attraverso una classificazione binaria.

In questa descrizione, così come nel paper originale, con il termine *frase* ci si riferisce a una porzione arbitraria di testo. BERT utilizza il tokenizer WordPiece e l'input che viene dato al modello BERT è una sequenza di token di 1 o 2 frasi. Gli embedding di BERT vengono costruiti non solo a partire dai token e dagli embedding posizionali, ma viene aggiunto un ulteriore livello di embedding, sommato agli altri due, detto *segment embedding*. Esso indica a quale frase di input appartiene un determinato token.

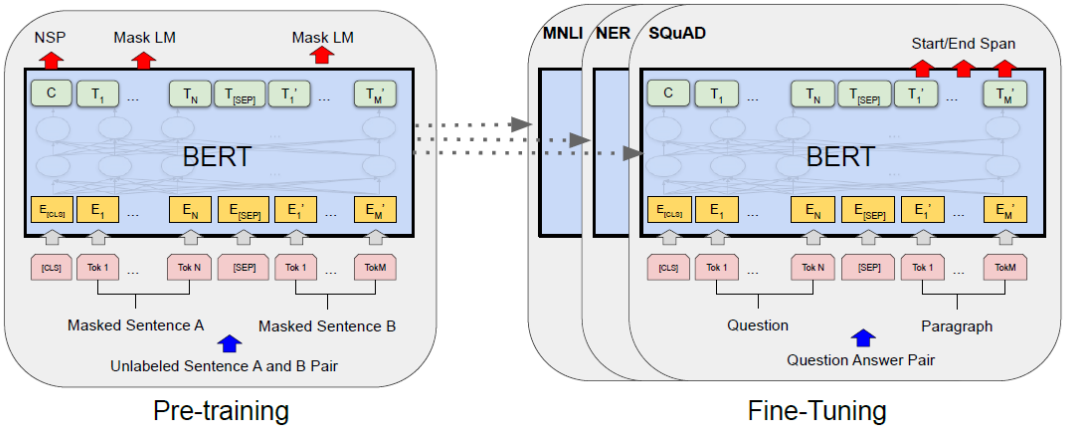


Figura 4.2. Schema dell'architettura di BERT. Il pre-training genera gli embedding tramite NSP e MLM, mentre il fine-tuning li ottimizza per il task specifico.

Nel pre-training di BERT, la funzione di loss è quindi costituita dalla somma di due funzioni di loss, una relativa al MLM e l'altra alla NSP. Entrambe sono delle cross-entropy loss per la classificazione. Il primo task consiste infatti nel classificazione di ogni token mascherato rispetto a tutti i token del dizionario, il secondo è invece un problema di classificazione binaria.

Nel lavoro di tesi è stata utilizzata la versione di BERT *uncased* che è cioè stata pre addestrata su un corpus testuale in cui tutte le parole sono convertite in lettere minuscole. Questa metodologia semplifica la complessità del modello. Dal modello BERT pre addestrato è stato poi eseguito un fine tuning di classificazione testuale, per il quale, come descritto al Sezione 2.3, è necessario aggiungere la *classification head*, il layer che consente di trasformare l'embedding che rappresenta la sequenza [CLS] in una distribuzione di probabilità sulle classi da predire.

4.2.3 RoBERTa

Nel 2019 è stato presentato il modello RoBERTa [21] (*Robustly Optimized BERT Approach*), un'evoluzione del modello BERT originale. I cambiamenti introdotti da RoBERTa riguardano variazioni nel processo di pre-training del modello, mentre l'architettura è la stessa di BERT. In particolare sono state introdotte le seguenti modifiche.

- *Dynamic masking.* Questa procedura consiste nel mascherare le parole per il MLM in maniera diversa per ogni epoch di addestramento. Nel modello BERT il campionamento delle parole mascherate avviene infatti in solo 10 modi diversi su 40 epoch di addestramento.
- *Rimozione NSP.* Gli autori hanno notato che dando in input una porzione di testo costituita da un insieme di frasi consecutive e rimuovendo il task di NSP si ottenevano risultati leggermente migliori rispetto a quelli ottenuti da BERT.
- *Aumento dimensione batch.* La dimensione dei batch di input è stata aumentata da 256 a 8000. Questa operazione è utile per la parallelizzazione del training.
- *Tokenizzazione BPE.* Per la tokenizzazione del testo è stata utilizzata la tecnica del Byte-Pair Encoding [33], nell'implementazione descritta in GPT-2 [28]. Questo ha generato un aumento della dimensione vocabolario da 30k a 50k unità.

Queste modifiche sono quindi state utilizzate per effettuare il pre training del modello a partire dalla versione large di BERT (che ha paramentri L = 24, H = 1024, A = 16). Nel training, è stato inoltre utilizzato un corpus testuale più ampio (pari a 160GB di testo) e un numero di step pari a 500k che, data la maggiore dimensione dei batch utilizzati, richiede un costo computazionale decisamente maggiore.

4.2.4 DistilBERT

Data l'elevata dimensione dei modelli precedenti, nel 2020 è stato sviluppato DistilBERT [30], un modello con l'obiettivo di ridurre il numero di parametri senza alterare le performance. Il training di DisilBERT è stato eseguito attraverso la *knowledge distillation*

[14], una tecnica che consiste nell'addestrare un modello più piccolo (detto studente) sulla base di un modello più grande (detto insegnante). Il processo di distillazione consiste nell'addestrare il modello studente sulla base di una cross-entropy loss rispetto all'output del modello insegnante (detta *loss di distillazione*). Per quanto riguarda DistilBERT, la loss complessiva è costituita da una combinazione lineare di funzioni di loss :

- loss di distillazione L_{ce} che valuta la discrepanza tra le probabilità di output del modello studente rispetto al modello insegnante
- loss di MLM L_{mlm}; è la stessa cross-entropy loss utilizzata in BERT per il task di MLM.
- loss di *cosine embedding* L_{cos}, data da 1 meno il coseno tra i 2 embedding, pensata per allineare i vettori di embedding tra BERT e DistilBERT.

Gli autori di DistilBERT riportano che il modello mantiene il 97% delle performance di BERT, con il 40% dei parametri in meno e un una riduzione del 60% del tempo di addestramento.

4.3 Modelli di summarization

I modelli di summarization richiedono un'architettura Transformer basata su encoder-decoder per avere sia in input che in output sequenze di testo. I modelli con architettura encoder-decoder sono infatti anche detti *sequence-to-sequence*. Per il lavoro sono stati utilizzati e messi a confronto i modelli *PEGASUS* e *BART*, il cui funzionamente è descritto nel seguito.

4.3.1 BART

Il modello BART (Bidirectional and Auto-Regressive Transformer), presentato nel 2019 nel paper [19], è costituito da un encoder bidirezionale e da un decoder autoregressivo unidirezionale (da sinistra a destra). Esso è sostanzialmente composto dall'unione di RoBERTa e del modello generativo GPT-2 [28]. A differenza di BERT, però, l'encoder di BART non presenta la rete feed-forward per generare la distribuzione di probabilità sui token mascherati. Gli embedding generati vengono infatti direttamente passati al decoder, in quanto, la loss function viene calcolata rispetto all'output del decoder. Nella versione base BART presenta 6 layer (iterazioni sull'intero blocco) sia per l'encoder che per il decoder, mentre nella versione large ne ha 12 per ognuno. Come per RoBERTa e GPT-2 viene utilizzato il Byte Pair Encoding per la creazione dei token.

Il modello è detto *denoising autoencoder* in quanto ha come obiettivo la ricostruzione del testo iniziale, dopo che esso ha subito diverse manipolazioni. BART può quindi essere addestrato per la generazione di testo sulla base di un task self-supervised.

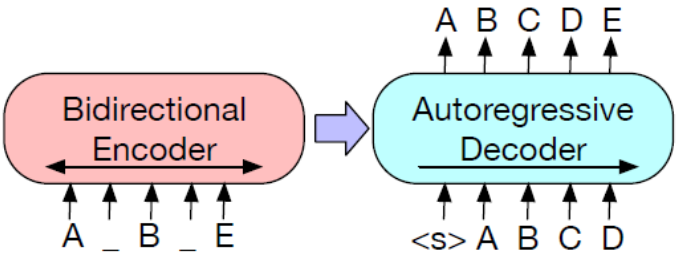


Figura 4.3. Schema dell'architettura di BART [19]. L'output dell'encoder bidirezionale è l'input del decoder autoregressivo, che viene utilizzato per produre il testo di output.

L'alterazione del testo di input avviene attraverso le seguenti operazioni.

- *Token mascherati.* Come avviene in BERT, alcuni token vengono sostituiti dal token speciale [MASK].
- *Token eliminati.* Alcuni token engono estratti casualmente e sono eliminati dal testo. In questo modo il modello deve capire in quali punti vi è del testo mancante.
- *Text infilling.* Questa procedura consiste nel campionare alcune sequenze testuali, la cui lunghezza di campionamento è variabile. In particolare, essa segue una distribuzione di Poisson con $\lambda = 3$. Le sequenze campionate vengono sostituite con un solo token speciale [MASK]. Questo consente al modello di essere allenato su quanti token mancano in un buco nel testo.
- *Permutazione delle frasi.* L'ordine delle frasi che compongono il testo viene modificato in modo casuale.
- *Rotazione del documento.* Viene campionato un token casualmente in modo uniforme e le parole del documento vengono ruotate in modo che esso cominci con il token estratto. Il modello deve così imparare a individuare l'inizio del testo.

L'input alterato viene quindi passato all'encoder, e l'obiettivo del decoder è ricostruire la sequenza di partenza in modo autoregressivo a partire da un token di inizio sequenza.

Questo tipo di pre training è generico, cioè il modello non viene pre addestrato per nessun task specifico. Si può poi effettuare il fine tuning sul task desiderato, come la text summarization. Per questo addestramento, l'input dell'encoder è il testo da riassumere, mentre la loss è la cross-entropy calcolata tra le probabilità di output del decoder e il vettore one-hot encoding del corrispettivo token del riassunto di riferimento.

4.3.2 PEGASUS

Il modello PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) presentato nel paper [43] è un modello con architettura sequence-to-sequence che nasce per lo specifico task di text summarization. A differenza di BART, infatti, il pre training non è generico, ma è sviluppato specificamente per la summarization. I token

vengono costruiti con il modello Unigram [16]. Gli obiettivi di pre training sono la Gap Sentences Generation (GSG) e il Masked Language Modeling (MLM).

La GSG consiste nell'individuare le frasi ritenute più importanti all'interno di un testo per poi eliminarle dall'input. Esse costituiranno infatti l'output di riferimento in fase di decoding. Il decoder dovrà quindi riuscire a ricostruire la frasi più importanti sulla base delle altre informazioni contenute nel testo. Le frasi più importanti vengono selezionate sulla base della metrica ROUGE-1 (illustrata nel paragrafo 5.2.1) misurata tra la frase da estrarre e il resto del testo.

Il MLM viene applicato in fase di encoding, applicando il masking del 15% delle parole, allo stesso modo in cui avviene in BERT. L'individuazione delle parole mascherate costituisce quindi l'obiettivo di output dell'encoder. A differenza del modello BART, la loss viene ottimizzata considerando sia l'output dell'encoder che quello del decoder. La figura seguente illustra l'architettura di PEGASUS e il funzionamento del pre training.

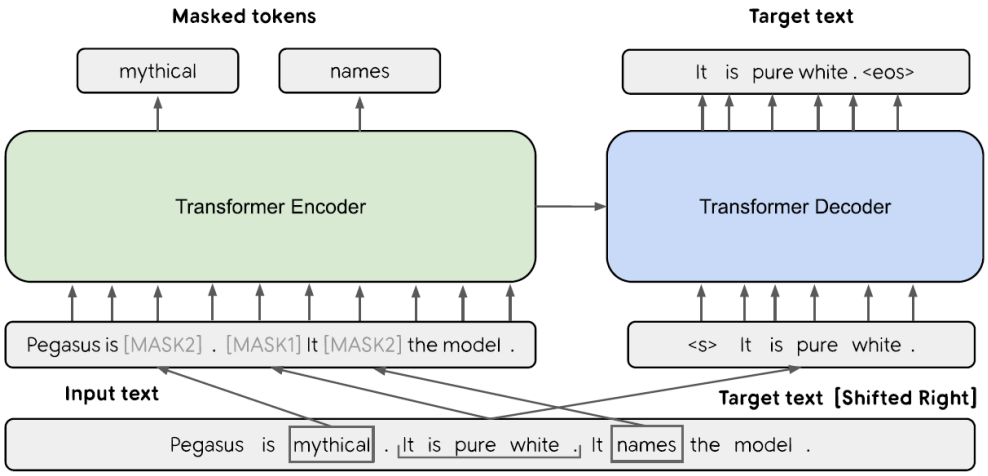


Figura 4.4. Schema dell'architettura di PEGASUS [43]. Il training avviene sia sulla base degli output dell'encoder che su quelli del decoder.

Dopo la fase di pre training su corpus generici, il modello è stato addestrato e valutati su 12 tipi di dataset diversi.

Capitolo 5

Esperimenti e risultati

Si presentano ora le metodologie con cui sono stati condotti gli esperimenti. In primo luogo saranno illustrate le metriche utilizzate per la classificazione e per la summarization. Successivamente si passerà alla descrizione della configurazione degli esperimenti, mostrando poi i risultati ottenuti.

5.1 Metriche per la classificatione

Gli algoritmi di classificazione vengono tipicamente valutati sulla base di *Accuracy*, *Precision*, *Recall* e *F1-score*. Esse sono definite nel contesto della classificazione binaria, a partire dalla matrice di confusione.

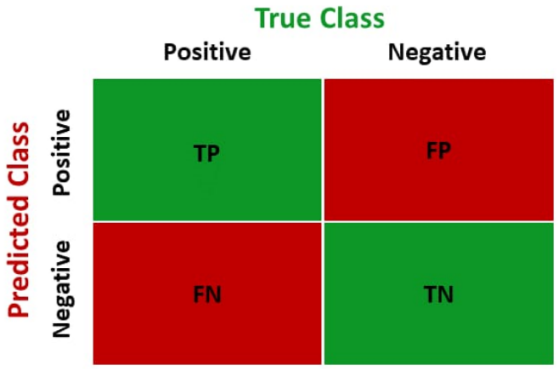


Figura 5.1. Matrice di confusione per la classificazione binaria.

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

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

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

$$\text{F1-score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Tali misure possono essere generalizzate nel contesto di classificazione non binaria. In particolare, l'accuracy, è definita come

$$\text{Accuracy} = \frac{TC_1 + TC_2 + ... + TC_N}{TOT},$$

dove C_1,..., C_N sono le N classi, e TOT è il numero di elementi totali. Le altre metriche devono essere generalizzate in maniera diversa, in quanto sono intrinsecamente definite per il caso binario. Si può considerare una classe alla volta e calcolare il valore della metrica considerando la classe in questione rispetto a tutte le altre, potendo così ricorrere alla definizione nel caso binario. Per avere una metrica complessiva, si può poi calcolare una media aritmetica o pesata sulla dimensione delle classi. In questo lavoro si è scelto di considerare le medie pesate.

Data la presenza della categoria di chiusura *non categorizzati*, è stato necessario definire metriche ad hoc per la valutazione del classificatore. Questa categoria deve infatti essere trattata in maniera particolare sia per la sua preponderanza rispetto alle altre, sia per il fatto che contiene elementi non omogenei tra loro. In particolare, sono state presi in considerazione quattro modi diversi di eseguire il calcolo delle metriche.

- Misura 1: valuta esclusivamente la capacità di distinguere gli elementi in *non categorizzati* dagli altri, applicando quindi le formule di accuracy, precision, recall e f1-score nel caso binario.
- Misura 2: valuta le performance del modello sul sottoinsieme di elementi che sono stati categorizzati in categorie vere, e che appartengono effettivamente a esse. In questo caso, così come nei successivi, va quindi utilizzata la generalizzazione delle metriche nel caso n classi. L'idea alla base di questo tipo di valutazione è comprendere le performance del modello in un mondo ideale, dove tutti i fondi hanno una categoria definita.
- Misura 3: valuta le performance del modello sul sottoinsieme di elementi che sono stati categorizzati in categorie vere, a prescindere dal fatto che appartengano ad esse nella realtà. In questo caso, si vuolo valutare la capacità del modello di classificare i fondi nelle categorie vere.
- Misura 4: valuta le performance complessive, considerando i *non categorizzabili* come una categoria come le altre. In questo modo si valuta la capacità generale del modello di eseguire una classificazione sia sulle categorie vere sia su quella di chiusura.

Il modello viene allenato per predire le categorie a prescindere che esse siano quelle vere o quella di chiusura. Le misure presentate entrano in gioco solamente nella valutazione delle performance, a partire dagli output del modello.

5.2 Metriche per la summarization

La text summarization richiede metriche ad hoc per la valutazione della bontà dei riassunti. Si vuole in generale valutare la similarità dei riassunti generati rispetto a quelli di riferimento. Le due tipologie di metriche utilizzate per questa analisi sono *ROUGE* e *BERTScore*.

5.2.1 ROUGE

Questa metrica è stata sviluppata nel 2004 [20] per valutare la bontà della summarization. La prima versione è detta ROUGE-N e misura quanti sono gli N-grammi (insiemi consecutivi di N parole) comuni tra testo di riferimento e testo di output rispetto al numero di N-grammi del testo di riferimento. Questo si traduce nella formula:

$$\text{ROUGE-N}_{\text{recall}} = \frac{\sum_{snt' \in R_S} \sum_{\text{N-gram} \in snt'} \text{Count}_{\text{match}}(\text{N-gram})}{\sum_{snt' \in R_S} \sum_{\text{N-gram} \in snt'} \text{Count}(\text{N-gram})},$$

dove snt' sono le frasi appartenenti ai riassunti di riferimento R_S, $\text{Count}_{\text{match}}$ è una funzione che vale 1 quando l'N-gramma in questione appartiene anche al riassunto di output e Count vale 1 per ogni N-gramma, in modo da contare gli N-grammi totali in R_S. ROUGE-N valuta quindi quanti degli N-grammi del testo di riferimento sono stati effettivamente usati nell'output, costituendo una forma di recall per dati testuali. È però comune utilizzare una versione di ROUGE che tiene conto anche della precision [36]:

$$\text{ROUGE-N}_{\text{precision}} = \frac{\sum_{snt \in P_S} \sum_{\text{N-gram} \in snt} \text{Count}_{\text{match}}(\text{N-gram})}{\sum_{snt \in P_S} \sum_{\text{N-gram} \in snt} \text{Count}(\text{N-gram})},$$

Essa, a differenza della recall, calcola il rapporto tra le gli N-grammi comuni e il numero di N-grammi del riassunto prodotto in output (P_S). La metrica a cui ci si riferisce con ROUGE-N è data dalla f1-score considerando la recall e la precision di ROUGE-N:

$$\text{ROUGE-N} = 2 \cdot \frac{\text{ROUGE-N}_{\text{recall}} \cdot \text{ROUGE-N}_{\text{precision}}}{\text{ROUGE-N}_{\text{recall}} + \text{ROUGE-N}_{\text{precision}}}.$$

Esiste un altro tipo di metrica ROUGE, sempre presentata nel paper originale [20], con l'obiettivo di misurare la somiglianza tra testi sulla base della sottostringa comune di lunghezza massima (LCS). Per poter comparare la LCS tra due testi X e Y di lunghezza diversa è necessario normalizzarla rispetto alla lunghezza del testo di riferimento (m) o del testo prodotto (n), ottenendo le due metriche seguenti:

$$\text{R}_{\text{LCS}} = \frac{\text{LCS}(X, Y)}{m}$$

$$\text{P}_{\text{LCS}} = \frac{\text{LCS}(X, Y)}{n}$$

Questi valori vengono combinati nel seguente modo per avere una misura unica:

$$\mathrm{F_{LCS}} = \frac{(1+\beta^2)\mathrm{R_{LCS}P_{LCS}}}{\mathrm{R_{LCS}} + \beta\mathrm{P_{LCS}}},$$

dove $\beta = \mathrm{P_{LCS}}/\mathrm{R_{LCS}}$. L'implementazione della metrica ROUGE-L, può essere eseguita in due modi. Il primo, detto ROUGE-L consiste nel calcolare $\mathrm{F_{LCS}}$ per ogni frase e farne la media aritmetica. Il secondo, denotato con ROUGE-Lsum consiste nel considerare direttamente tutta la sequenza nel calcolo di $\mathrm{F_{LCS}}$.

5.2.2 BERTScore

Una limitazione delle metriche ROUGE è che esse valutano la somiglianza tra testi solamente sulla base delle parole identiche. Nel caso in cui i testi presentino parole diverse, ma semanticamente molto simili, le metriche ROUGE fornirebbero comunque risultati molto bassi, nonostante la somiglianza semantica tra i testi. Per questo motivo, nel 2020 è stata sviluppata BERTScore [44], una metrica che valuta la somiglianza semantica tra testi sulla base dei word embedding del modello BERT. In generale, la somiglianza tra due vettori x e y può essere valutata attraverso la cosine similarity:

$$\text{cosine similarity} = \frac{x^T y}{\|x\|\|y\|}.$$

Essa è semplicemente il valore del coseno dell'angolo tra i due vettori. L'idea di BERTScore è di valutare la somiglianza tra due sequenze attraverso la somma delle cosine similarity tra le coppie di embedding da cui sono costituite. Questo può essere svolto in due modi diversi.

Il primo è di considerare gli embedding dei token appartenenti alla sequenza di riferimento e considerare, per ognuno di essi, la cosine similarity massima con gli embedding della sequenza prodotta. In questo modo ogni token viene accoppiato con il token di output più semanticamente vicino. Viene poi calcolata una media aritmetica su tutti i token della sequenza di riferimento. Questo si riassume nella formula

$$\mathrm{R_{BERT}} = \frac{1}{|R_S|} \sum_{x \in R_S} \max_{y \in P_S} \frac{x^T y}{\|x\|\|y\|},$$

dove R_S è la sequenza di riferimento e $|R_S|$ è la sua lunghezza.

Il secondo modo consiste nell'eseguire un calcolo simile, ma a partire dagli embedding del testo di output rispetto a quelli del testo di riferimento:

$$\mathrm{P_{BERT}} = \frac{1}{|P_S|} \sum_{y \in P_S} \max_{x \in R_S} \frac{x^T y}{\|x\|\|y\|}.$$

Queste metriche, che sono una forma di recall e di precision, possono essere combinate nella f1-score:

$$\mathrm{F1_{BERT}} = 2 \cdot \frac{\mathrm{P_{BERT}} \cdot \mathrm{R_{BERT}}}{\mathrm{P_{BERT}} + \mathrm{R_{BERT}}},$$

BERTScore con idf

Le metriche appena mostrate vengono calcolate sulla base delle medie aritmetiche delle cosine similarity, in questo modo, tutti gli embedding hanno la stessa importanza. Esse possono però essere calcolate in maniera leggermente diversa, eseguendo una media pesata sulla base della *inverse document frequency (idf)* calcolata sui token, in modo da attribuire un peso diverso in base all'importanza del token nella sequenza. La idf per la parola w è definita nel seguente modo:

$$\text{idf}(w) = -\log\left(\frac{1}{M}\sum_{i=1}^{M}\mathbb{I}[w \in S^{(i)}]\right),$$

dove M è il numero di sequenze, $S^{(i)}$ è la i-esima sequennza e $\mathbb{I}$ è la funzione indicatrice. La recall e la precision di BERT possono quindi essere calcolate nel seguente modo:

$$\text{R}_{\text{BERT\_idf}} = \frac{\sum_{x\in R_S}\text{idf(x)}\cdot\max_{y\in P_S}x^T y}{\sum_{x\in R_S}\text{idf(x)}}$$

$$\text{P}_{\text{BERT\_idf}} = \frac{\sum_{y\in P_S}\text{idf(y)}\cdot\max_{x\in R_S}x^T y}{\sum_{y\in P_S}\text{idf(y)}},$$

dalle quali si può ricavare la formula per la relativa $\text{F}_{\text{BERT\_idf}}$.

Il valore di tutte le metriche BERTScore sarà un numero compreso tra -1 e 1, dato che deriva da una media del valore di coseni. Inoltre, come viene riportato nel paper [44], i valori tendono a concentrarsi in un raggio più piccolo. Per questi motivi gli autori hanno operato una trasformazione in modo che i valori di BERTScore sia ben distribuiti tra 0 e 1. In particolare, la metrica è stata calcolata su una grande quantità di frasi accoppiate in modo casuale, in modo da ottenere un valore b che rappresenti un estremo inferiore empirico. Il valore di R_{BERT} è stato poi normalizzato nel seguente modo:

$$\hat{\text{R}}_{\text{BERT}} = \frac{\text{R}_{\text{BERT}} - b}{1-b}. \tag{5.1}$$

Oltre a valutare la vicinanza semantica tra parole simili, un altro vantaggio di BERTScore è di non avere un criterio di valutazione basato su una struttura rigida dell'ordine delle parole, a differenza degli n-grammi di ROUGE.

5.3 Esperimenti summarization

In questa sezione si riportano le modalità con cui è stata eseguita la summarization e i risultati che si sono ottenuti. Il codice è stato sviluppato grazie alla libreria *Transformers* di *Huggingface*, che, oltre a distribuire i modelli, consente di utilizzare un'interfaccia per la loro configurazione. In particolare, sono stati utilizzati e messi a confronto i due modelli seguenti.

- *bart-large-cnn*, la versione di BART che ha eseguito il fine tuning sul task di summarization con sul dataset CNN/Dailymail [32], che racchiude testi di notizie con i relativi highlights.

- *pegasus-cnn_dailymail*, la versione di PEGASUS che ha subito l'addestrmento sul dataset CNN/Dailymail.

Per semplicità, ci si riferirà a questi modelli con *BART* e *PEGASUS* rispettivamente. Essi sono stati scaricati dall'Huggingface Hub [41].

La prima operazione consiste nella tokenizzazione dei testi di input, che avviene con il tokenizer relativo al modello utilizzato (invocato attraverso il comando *AutoTokenizer.from_pretrained(nome_modello)*). Con entrambi i modelli, i token prodotti nei dati di training sono al massimo 940 per quanto riguarda i testi originali e 112 per i riassunti di riferimento. Questo motiva la scelta del parametro *max_length* per la tokenizzazione dei testi che è stato posto pari a 1024 per quanto riguarda i testi di input e pari a 128 per i testi target (i riassunti).

Dopo la costruzione dei token si passa all'addestramento del modello. Un parametro importante in questa fase è il numero di epoch. Per scegliere questo parametro si sono osservati i valori di training loss e validation loss durante le epoch di addestramento. Nel seguente grafico ne è riportato l'andamento per il modello BERT.

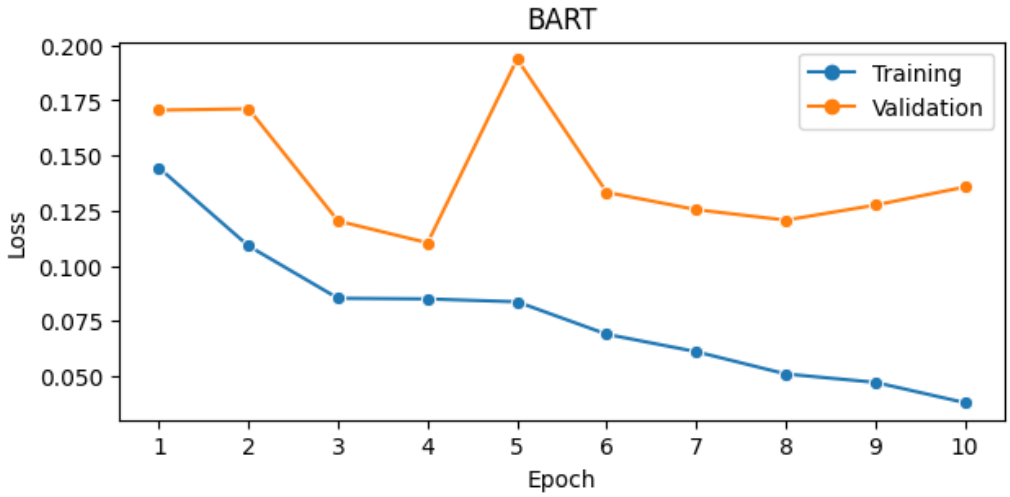


Figura 5.2. Grafico delle loss del modello BART al variare delle epoch di addestramento. Dalla 4 in poi si verifica l'overfitting.

Si può notare che la training loss continua a scendere allo scorrere delle epoch, mentre la validation loss raggiunge il suo minimo alla epoch 4. Dopo aver registrato un picco alla epoch 5, essa è crescente dalla 6 in poi. Questo significa che, se il modello viene addestrato con un numero troppo elevato di epoch, si verifica il fenomeno dell'*overfitting*. Esso consiste in un eccessivo adattamento del modello ai dati di training, che che causa

un aumento della loss sui dati di validation. Il numero di epoch di addestramento scelto è perciò 4.

Nel seguente grafico si mostra invece l'andamento di training e validation loss ottenuto dal modello PEGASUS.

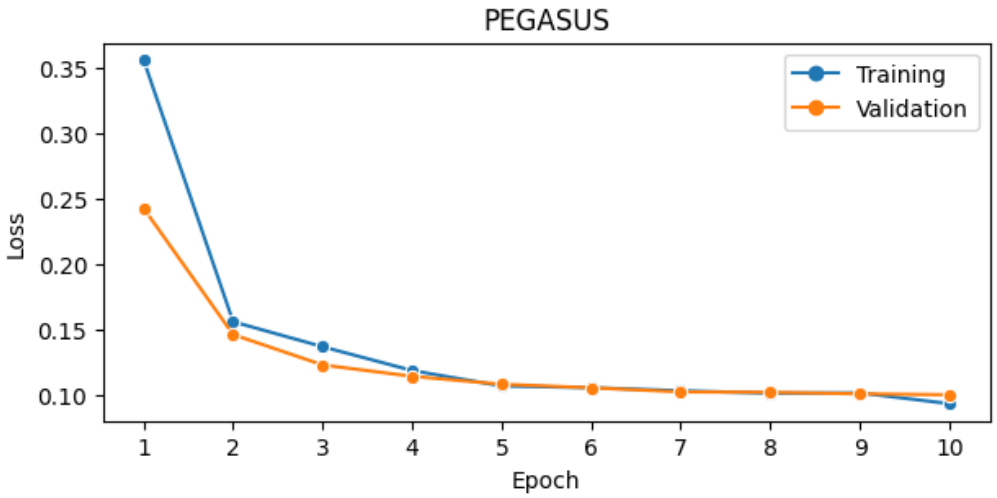


Figura 5.3. Grafico delle loss del modello PEGASUS al variare delle epoch di addestramento. Non si verifica overfitting.

In questo caso entrambe le loss calano bruscamente alla seconda epoch per poi continuare entrambe a calare più lentamente. In questo caso non si verifica overfitting e si è scelto un numero di epoch pari a 10.

L'oggetto *TrainingArguments* della libreria Transformers racchiude tutti gli iperparametri per il training del modello. Si riporta che per tutti i modelli Transformer utilizzati, il training è stato eseguito utilizzando l'ottimizzatore di default AdamW [22]. Sono stati adottati i seguenti valori per entrambi i modelli:

TrainingArguments(output_dir=dir, num_train_epochs=num_epochs, warmup_steps=500, per_device_train_batch_size=1, per_device_eval_batch_size=1, weight_decay=0.01, logging_steps=10, push_to_hub=False, evaluation_strategy='epoch', save_steps = 1e6, gradient_accumulation_steps=16)

L'*evaluation_strategy* consente di valutare l'andamento delle loss dopo ogni epoch, mentre il valore *save_steps* posto a 10^6 serve a non effettuare salvataggi del modello intermedi, per evitare di occupare troppa memoria. Gli altri parametri numerici sono stati scelti su valori standard, in riferimento a [36]. Dal momento che i modelli sono molto pesanti da addestrare, la dimensione del batch è pari a 1. Per questo motivo viene applicata la gradient accumulation, in modo da calcolare il gradiente sulla base di 16 campioni (Appendice A).

In fase di generazione testuale sono stati invece utilizzati i parametri $num\_beams = 8$ e $length\_penalty = 0.8$.

5.3.1 Risultati summarization

Si mostrano ora i risultati ottenuti dai due modelli PEGASUS e BART. Per quanto riguarda le metriche ROUGE si sono ottenuti i seguenti valori.

	rouge1	rouge2	rougeL	rougeLsum
google/pegasus-cnn_dailymail (10 ep.)	0.83	0.73	0.83	0.83
facebook/bart-large-cnn (4 ep.)	0.82	0.72	0.81	0,81

Le metriche registrano valori discretamente alti, cioè con entrambi i modelli gli output non si discostano eccessivamente dai testi di riferimento. PEGASUS sembra mostrare valori leggermente migliori, anche se a scapito di un maggior tempo di addestramento richiesto, a causa delle 10 epoch.

La metrica BERTScore registra invece i seguenti valori.

	R_{BERT}	P_{BERT}	$F1_{BERT}$
google/pegasus-cnn_dailymail (10 ep.)	0.90	0.92	0.91
facebook/bart-large-cnn (4 ep.)	0.92	0.94	0.93

In questo caso i valori sono sempre alti, ma modello BART mostra performance leggermente superiori. Si mostrano infine i valori di BERTScore con idf.

	R_{BERT_idf}	P_{BERT_idf}	$F1_{BERT\_idf}$
google/pegasus-cnn_dailymail (10 ep.)	0.82	0.90	0.86
facebook/bart-large-cnn (4 ep.)	0.90	0.92	0.91

I valori di BERT sono migliori di quelli di PEGASUS, soprattutto per quanto riguarda recall e f1-score.

Si mostra ora un esempio di come il testo originale è stato riassunto da PEGASUS e da BART, confrontando gli output con il riassunto di riferimento.

Testo originale:
'Nell'ambito dell'esposizione ai Mercati emergenti, il comparto può investire fino al 20% del patrimonio netto in azioni emesse da società con sede principale nella Cina continentale (comprese le Azioni A cinesi). Il comparto è un prodotto che promuove caratteristiche ambientali o sociali o una combinazione di tali caratteristiche ai fini dell'articolo 8 del Regolamento relativo all'informativa sulla sostenibilità nel settore finanziario. Il comparto punta a generare una crescita del capitale investendo principalmente in titoli azionari emessi da società in tutto il mondo (compresi i Mercati emergenti) con notorietà del marchio leader/premium secondo il parere del Gestore e/o che offrono prodotti e/o servizi di lusso, premium e prestige, o che conseguono la maggior parte dei loro ricavi consigliando, fornendo, realizzando o finanziando le suddette attività.'

Riassunto riferimento:
'Il fondo investe in azioni in società con sede in tutto il mondo, specialmente in Mercati sviluppati ed emergenti. Il fondo investe principalmente nel settore Consumi discrezionali,

in aziende All Cap. Il fondo ha uno stile Growth & Value.'

Riassunto generato da BART:
'Il comparto deterrà prevalentemente azioni in società con sede in tutto il mondo, specialmente in Mercati sviluppati ed emergenti, in aziende Large & Mid Cap di tipo Growth & Value.'

Riassunto generato da PEGASUS:
'Il comparto deterr prevalentemente azioni in societ con sede in tutto il mondo, specialmente in Mercati sviluppati ed emergenti, in societ Large & Mid Cap di tipo Value.'

Si può notare come, sia con BERT che con PEGASUS, i riassunti siano testi di senso compiuto. In questo esempio ci sono piccole differenze tra i due modelli, solamente BART ha individuato il corretto stile del fondo. In generale, ad esclusione di alcune informazioni come il settore, i riassunti generati sono simili a quello di riferimento. Si può inoltre vedere come PEGASUS sembra mostrare errori di codifica per le parole con lettere accentate.

Sulla base di questi confronti si può in generale affermare come, al di là del valore delle metriche, i riassunti generati siano qualitativamente di discreta qualità.

5.4 Esperimenti classificazione

La classificazione testuale è stata effettuata con i modelli Transfomer *bert_base_uncased*, *distilbert_base_uncased* e *roberta_base*, attraverso la libreria di Huggingface [41]. Inoltre, è stato utilizzato il modello *Doc2vec-Dbow*, tramite la libreria Gensim [29], in combinazione con il modello SVC di Scikit-learn [27]. Si illustra nel seguito la configurazione di iperparametri utilizzata con Doc2vec e con i Transformer.

Configurazione Doc2vec

Per regolare alcuni iperparametri del modello Doc2vec, sono state provate diverse configurazioni, analizzando il comportamento del classificatore SVC (con iperparametro $C = 1$) con i vettori prodotti. In particolare si è scelto di valutare la media dei valori delle accuracy delle 4 metriche considerate, al variare dei parametri *epochs* e *vector_size*. Sono stati utilizzati i seguenti valori:

- *epochs*: 20, 50, 100, dove è stato usato lo stesso valore sia per le epoch di addestramento che per quelle di inferenza.
- *vector_size*: 20, 50, 100. Questi valori rappresentano la dimensione dello spazio vettoriale in cui vengono creati gli embedding.

Si sono analizzati i risultati sul validation set, con le diverse combinazioni dei valori appena riportati, avendo così 9 configurazioni diverse. Gli altri parametri utilizzati sono i seguenti:

[*dm=0, negative=5, hs=0, min_count=15, sample = 0, workers=cores, alpha=0.025, min_alpha=0.001*]

Il parametro $dm = 0$ indica l'utilizzo dell'architettura Dbow; min_count è il numero per il quale le parole che compaiono un numero minore di volte vengono ignorate; $sample = 0$ indica che non viene eseguito alcun tipo di sottocampionamento sulle parole più frequenti; $alpha$ è il learning rate iniziale e min_alpha è il learning rate finale (trai quali avviene descrescita lineare); infine $hs = 0$ e $negative = 5$ indicano l'utilizzo della tecnica del negative sampling, illustrata in [25], utilizzata per calcolare un'approssimazione dei valori della funzione softmax (Appendice A)

Si mostrano i valori delle accuracy (arrodondate al secondo decimale) ottenute con la SVM a seconda delle diverse configurazioni del Doc2vec (con il formato *vector_size*|*epoch*) e sui diversi input testuali (testi originali, riassunti con PEGASUS e riassunti con BART):

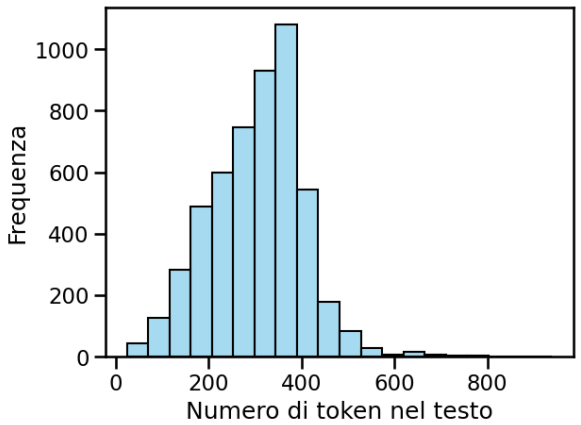
	20|20	20|50	20|100	50|20	50|50	50|100	100|20	100|50	100|100
Testi	**0.43**	0.41	0.39	**0.43**	0.42	0.41	0.42	0.40	0.36
Sum P.	0.52	0.55	**0.56**	0.52	**0.56**	**0.56**	0.52	**0.56**	**0.56**
Sum B.	0.49	**0.52**	0.51	0.50	**0.52**	**0.52**	0.50	0.51	**0.52**

A parità di prestazioni sono state scelte le configurazioni meno dispendiose, ovvero:

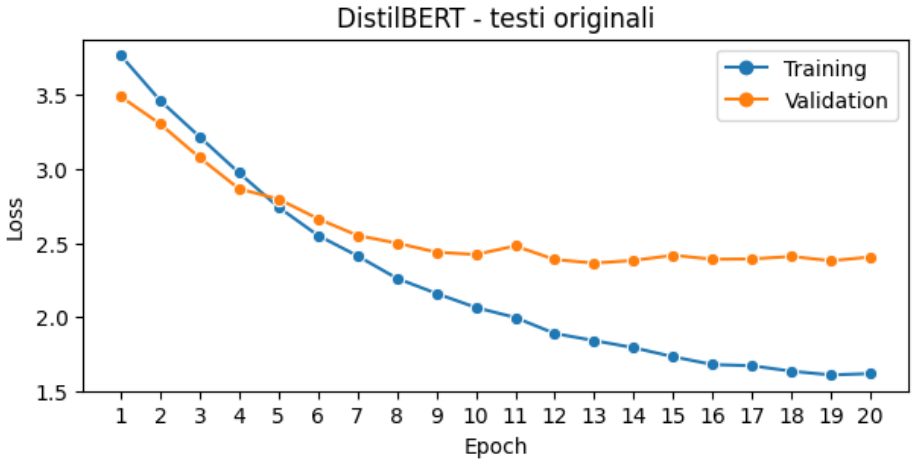
- Testo originale: $vector\_size = 20$, $epochs = 20$.
- Riassunti Pegasus: $vector\_size = 20$, $epochs = 100$.
- Riassunti Bart: $vector\_size = 20$, $epochs = 50$.

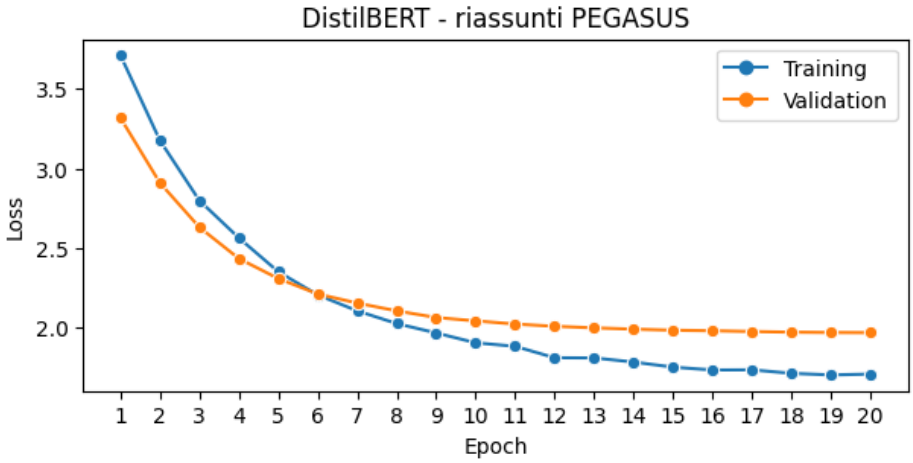
Configurazione Distilbert

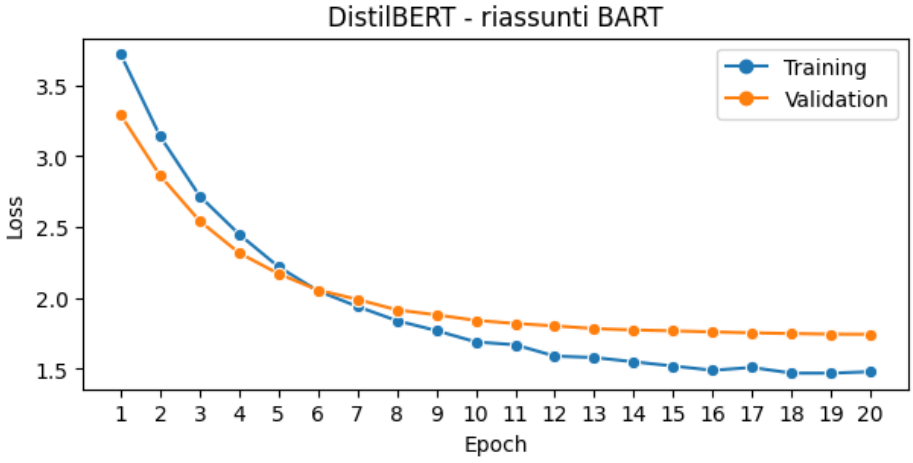
Come per la summarization, si è analizzato il comportamento dei modelli in base al valore della loss al crescere delle epoch di addestramento (fino a 20). L'input del modello DistilBERT, deve avere un numero di token non superiore a 512. Per questo motivo, i testi originali più lunghi di questo valore sono stati troncati. Dal seguente istogramma si può notare che essi sono una quota minoritaria.



Il loro troncamento non inciderà eccessivamente sull'informazione del testo che viene passata al modello. I valori di training e validation loss sono rappresentati nei seguenti grafici, a seconda dell'input utilizzato.


DistilBERT - testi originali
Training
Validation
Loss
Epoch

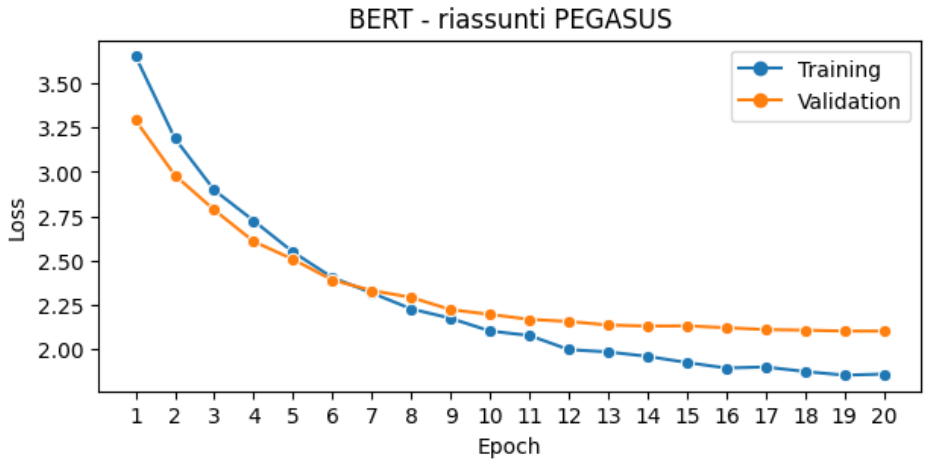

DistilBERT - riassunti PEGASUS
Training
Validation
Loss
Epoch

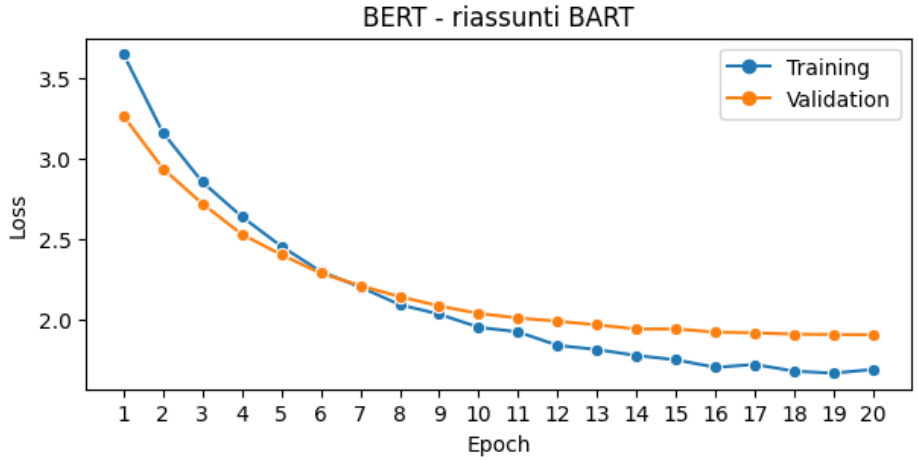

DistilBERT - riassunti BART
Training
Validation
Loss
Epoch

Per quanto riguarda la classificazione con i riassunti generati, si nota che la validation loss continua a calare, seppur lentamente, per cui, in entrambi i casi, si è preso 20 come numero di epoch. Con il testo originale si può vedere invece come la validation loss tenda a restare costante dalla epoch 10 in poi, numero che è stato scelto per eseguire gli esperimenti.

Configurazione BERT

Anche per il modello BERT sono state eseguite le analisi sull'andamento della loss su 20 epoch. A differenza di DistilBERT, non sono stati eseguiti esperimenti sui testi originali, dal momento che con un numero di token pari a 512 le risorse di Google Colab non sono sufficienti per eseguire il modello. Non sono stati effettuati esperimenti troncando gli input a un numero di token inferiore in quanto si è ritenuto che i testi sarebbero risultati troppo poco rappresentativi.





In entrambi i casi la validation loss continua a descrescere lentamente, per cui si è scelto di prendere 20 come numero di epoch.

Oltre alle epoch, i parametri di training, contenuti nell'oggetto TrainingArguments, sono gli stessi per tutti i modelli Transformer utilizzati per la classificazione. Essi sono i seguenti:

TrainingArguments(label_names = ['labels'], output_dir=dir, num_train_epochs=num_epochs, learning_rate=2e-5, per_device_train_batch_size=batch_size, per_device_eval_batch_size=batch_size, weight_decay=0.01, evaluation_strategy='epoch', disable_tqdm=False, logging_steps=logging_steps, push_to_hub=False, log_level='error', save_strategy = 'no').

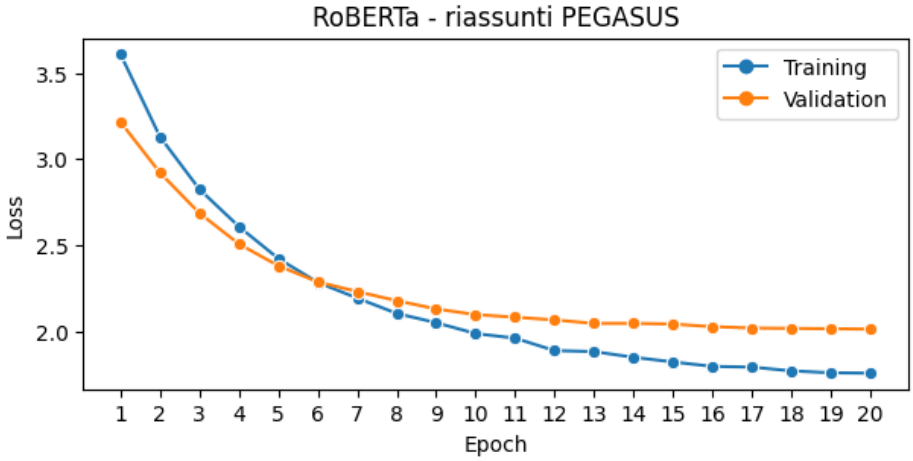
Anche in questo caso sono stati scelti valori standard, in riferimento a [36]. In particolare *batch_size* è stato scelto pari a 32 e *logging_steps* è definito come

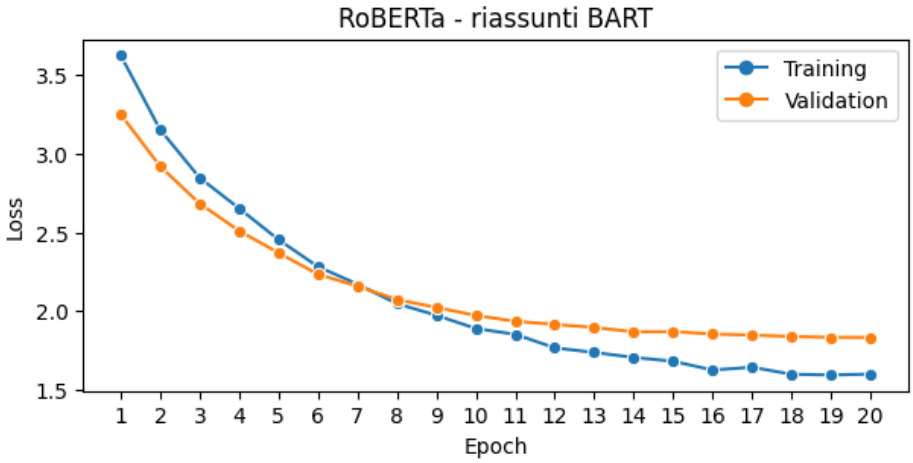
len(data['train'])//batch_size,

dove // è l'operatore di divisione intera.

Configurazione RoBERTa

Per il modello RoBERTa valgono le stesse considerazioni fatte per BERT sui limiti computazionali di Colab, per cui si sono eseguiti gli esperimenti solamente sui riassunti generati. Si mostrano i grafici sull'andamento delle loss.





Anche in questo caso, non essendosi presentati fenomeni di overfitting, si è scelto 20 come numero di epoch.

5.4.1 Risultati classificazione

Si mostrano ora i risultati ottenuti nella classificazione dai diversi modelli, evidenziando il confronto tra i risultati ottenuti con i testi originali e quelli con i riassunti generati da PEGASUS e BART. L'obiettivo dei riassunti è di concentrare le informazioni che sono rilevanti per individuare le categorie, per cui ci si aspetta che le misure 2 e 3 con i riassunti diano risultati migliori rispetto a quelli con i testi originali. Per queste due misure è rilevante anche il supporto, dato che esse vengono calcolate su una porzione degli output.

Doc2vec + SVM

Il modello di classificazione SVM basato sugli embedding Doc2vec ha ottenuto i seguenti risultati.

Doc2vec + SVM	Misura 1			
	Accuracy	Precision	Recall	F1-score
Testo originale	0.57	0.69	0.57	0.56
Riassunti PEGASUS	0.59	0.66	0.59	0.59
Riassunti BART	0.60	0.64	0.60	0.60

Doc2vec + SVM	Misura 2				
	Accuracy	Precision	Recall	F1-score	Supporto
Testo originale	0.37	0.32	0.37	0.33	267
Riassunti PEGASUS	0.58	0.44	0.58	0.48	334
Riassunti BART	0.63	0.52	0.63	0.56	378

Doc2vec + SVM	Misura 3				
	Accuracy	Precision	Recall	F1-score	Supporto
Testo originale	0.30	0.23	0.30	0.25	322
Riassunti PEGASUS	0.45	0.28	0.45	0.33	430
Riassunti BART	0.47	0.30	0.47	0.35	513

Doc2vec + SVM	Misura 4			
	Accuracy	Precision	Recall	F1-score
Testo originale	0.42	0.31	0.42	0.33
Riassunti PEGASUS	0.47	0.32	0.47	0.37
Riassunti BART	0.47	0.35	0.47	0.39

Dai risultati si può vedere che i riassunti mostrano miglioramenti in tutte le metriche, soprattutto nelle misure 2 e 3, confermando le ipotesi. I riassunti generati da BERT hanno risultati migliori, sia nei valori delle metriche, che neI valori del supporto.

DistilBERT

Il modello DistilBERT ha ottenuto i seguenti risultati.

DistilBERT	Misura 1			
	Accuracy	Precision	Recall	F1-score
Testo originale	0.63	0.73	0.63	0.62
Riassunti PEGASUS	0.62	0.67	0.62	0.63
Riassunti BART	0.65	0.67	0.65	0.65

DistilBERT	Misura 2				
	Accuracy	Precision	Recall	F1-score	Supporto
Testo originale	0.33	0.18	0.33	0.21	324
Riassunti PEGASUS	0.54	0.41	0.54	0.44	383
Riassunti BART	0.59	0.49	0.59	0.52	438

DistilBERT	Misura 3				
	Accuracy	Precision	Recall	F1-score	Supporto
Testo originale	0.28	0.14	0.28	0.17	373
Riassunti PEGASUS	0.41	0.25	0.41	0.29	497
Riassunti BART	0.44	0.30	0.44	0.34	580

DistilBERT	Misura 4			
	Accuracy	Precision	Recall	F1-score
Testo originale	0.43	0.28	0.43	0.32
Riassunti PEGASUS	0.46	0.33	0.46	0.37
Riassunti BART	0.48	0.38	0.48	0.42

Anche con il modello DistilBERT, con i riassunti si evidenziano risultati migliori. Per quanto riguarda la misura 1, esso mostra risultati leggermente migliori di Doc2vec, mentre

le misure 2 e 3 sono leggermente peggiori. I valori del supporto sono però più elevati. Questo si spiega con il fatto che i fondi appartenenti a categorie vere sono individuati in maniera più corretta rispetto al modello Doc2vec (dato che la misura 1 è più elevata). Di conseguenza, il supporto è maggiore perché vi sono meno fondi di categorie vere classificati per errore nella categoria di chiusura. Un maggiore supporto implica maggiore diversità tra fondi, che può essere una delle cause delle prestazione peggiori sulle metriche 2 e 3 rispetto a quelle con Doc2vec. La misura 4 assume invece valori simili a Doc2vec.

BERT

Come anticipato, per motivi computazionali, con il modello BERT non è possibile avere i risultati per i testi originali. Si mostra perciò il confronto tra i riassunti generati.

BERT	Misura 1			
	Accuracy	Precision	Recall	F1-score
Riassunti PEGASUS	0.64	0.68	0.64	0.65
Riassunti BART	0.64	0.67	0.64	0.65

BERT	Misura 2				
	Accuracy	Precision	Recall	F1-score	Supporto
Riassunti PEGASUS	0.50	0.37	0.50	0.41	403
Riassunti BART	0.54	0.42	0.54	0.46	434

BERT	Misura 3				
	Accuracy	Precision	Recall	F1-score	Supporto
Riassunti PEGASUS	0.39	0.24	0.39	0.28	516
Riassunti BART	0.41	0.25	0.41	0.30	573

BERT	Misura 4			
	Accuracy	Precision	Recall	F1-score
Riassunti PEGASUS	0.46	0.33	0.46	0.37
Riassunti BART	0.47	0.36	0.47	0.39

I riassunti BART performano sempre meglio di quelli PEGASUS e i risultati sono in generale paragonabili a quelli di DistilBERT o leggermente peggiori.

RoBERTa

Come per il modello BERT si mostrano le metriche ottenute con i riassunti generati.

RoBERTa	Misura 1			
	Accuracy	Precision	Recall	F1-score
Riassunti PEGASUS	0.64	0.67	0.64	0.64
Riassunti BART	0.66	0.68	0.66	0.66

RoBERTa	Misura 2				
	Accuracy	Precision	Recall	F1-score	Supporto
Riassunti PEGASUS	0.53	0.40	0.53	0.43	411
Riassunti BART	0.59	0.49	0.59	0.52	456

RoBERTa	Misura 3				
	Accuracy	Precision	Recall	F1-score	Supporto
Riassunti PEGASUS	0.40	0.25	0.40	0.29	537
Riassunti BART	0.45	0.29	0.45	0.34	602

RoBERTa	Misura 4			
	Accuracy	Precision	Recall	F1-score
Riassunti PEGASUS	0.46	0.35	0.46	0.38
Riassunti BART	0.49	0.39	0.49	0.42

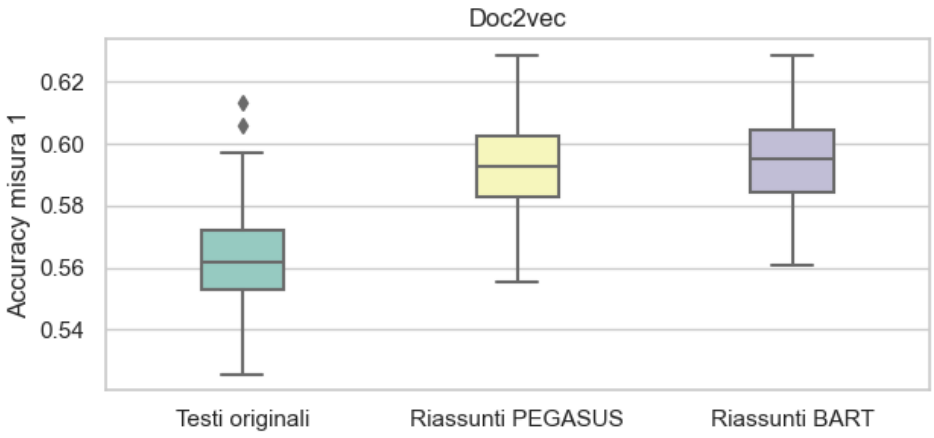
I risultati sono tendenzialmente migliori di quelli degli altri modelli Transformer e si avvicinano ai valori del Doc2vec. Rispetto a quest'ultimo, però, i valori del supporto sono più elevati. Questo significa che vi sono meno fondi che vengono inseriti per sbaglio nella categoria di chiusura e quindi vi è una maggiore variabilità di dati per le metriche 2 e 3.

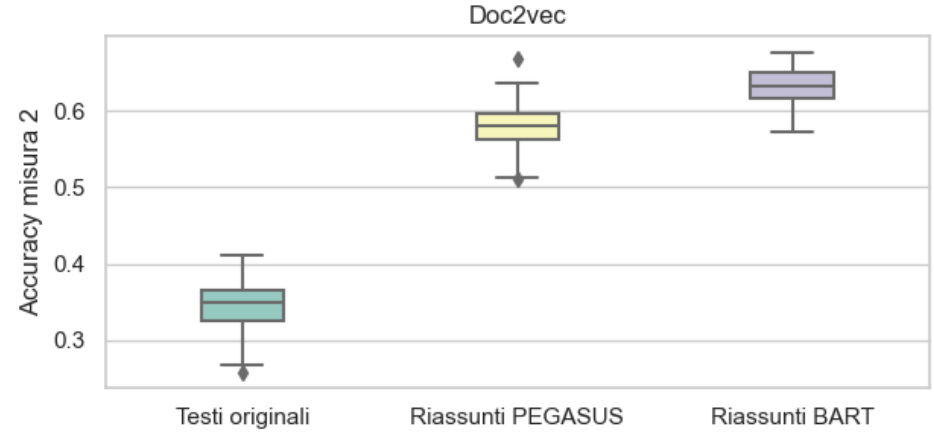
5.4.2 Validazione con Bootstrap

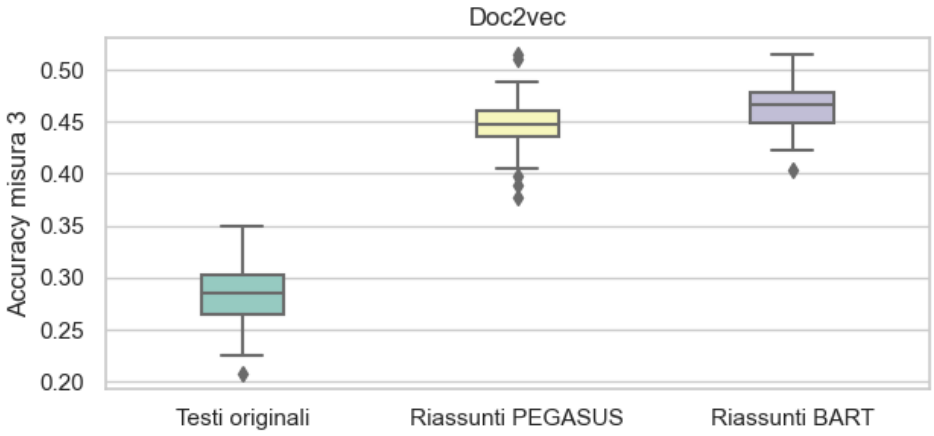
I risultati riportati fanno riferimento a una singola performance dei modelli. Purtroppo non è stato possibile validare i risultati ottenuti sulla base di statistiche su diversi campioni di metriche di output, dal momento che non sono disponibili altri dati di test. Non si sono nemmeno potuti effettuare nuovi cicli di addestramento per ottenere una serie di modelli equivalenti, sui quali valutare la metriche. Infatti momento che il fine tuning dei Transformer è molto dispendioso in termini di risorse computazionali e richiederebbe troppo tempo per essere eseguito un numero consistente di volte.

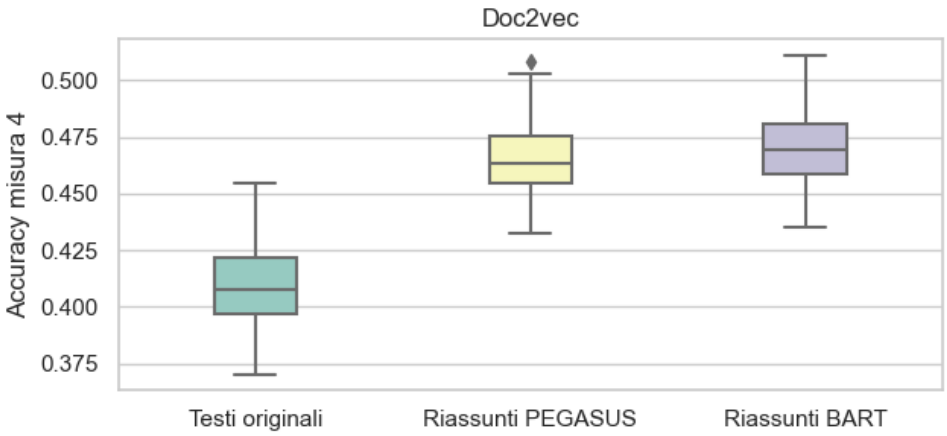
Per validare i risultati, mostrando la robustezza della metodologia proposta, si è scelto, perciò, di applicare la tecnica Bootstrap sui dati di test, per ottenere diversi valori delle accuracy. È stato effettuato per 100 volte un campionamento con reimbussolamento di dimensione pari a quella dei dati di test. Con i risultati ottenuti è stata stimata la distribuzione dei valori delle accuracy delle diverse metriche attraverso dei boxplot. In particolare, sono state messe a confronto le distribuzioni ottenute con i testi e quelle ottenute con i riassunti, con i modelli DistilBERT e Doc2vec con SVM.

Si mostrano i boxplot relativi alla distribuzione delle accuracy per il modello Doc2vec con SVM.


Doc2vec
Accuracy misura 1
0.62
0.60
0.58
0.56
0.54
Testi originali
Riassunti PEGASUS
Riassunti BART

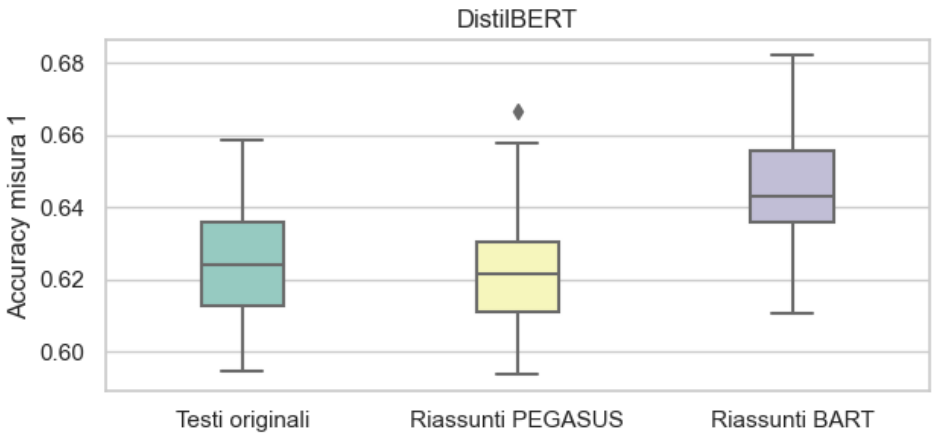

Doc2vec
Accuracy misura 2
0.6
0.5
0.4
0.3
Testi originali
Riassunti PEGASUS
Riassunti BART

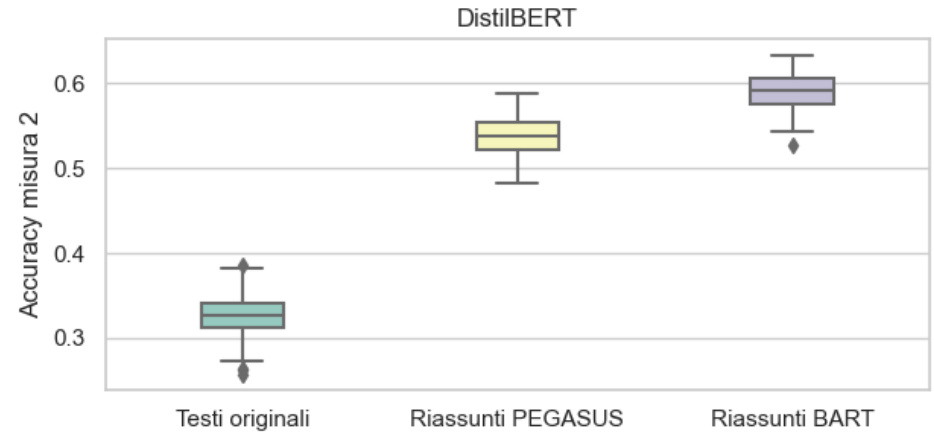

Doc2vec
Accuracy misura 3
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Testi originali
Riassunti PEGASUS
Riassunti BART

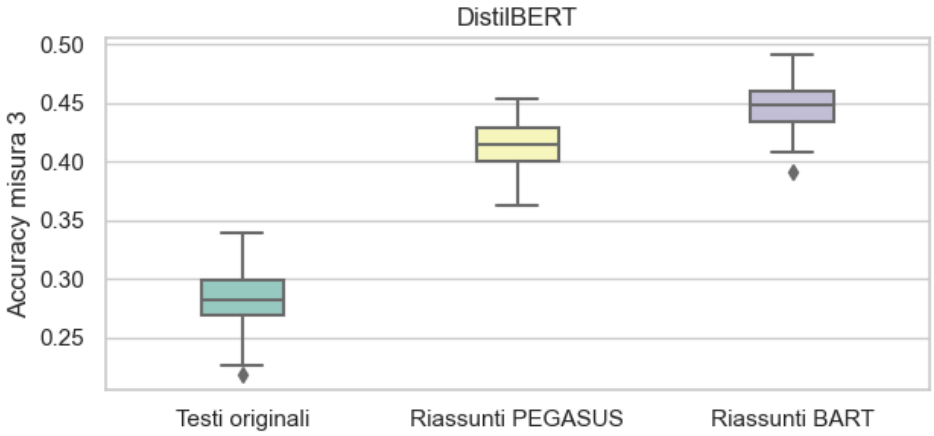


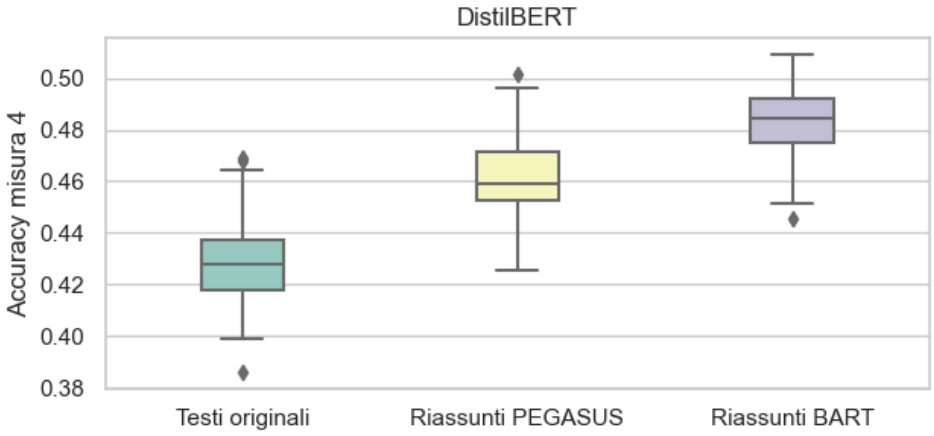
Per quanto riguarda la misura 1, i miglioramenti della accuracy utilizzando i riassunti sono poco significativi. Nonostante il lieve miglioramento medio, le distribuzioni si sovrappongono parzialmente. I miglioramenti più netti si possono invece notare nelle misure 2 e 3, per le quali tutti i campioni bootstrap relativi ai riassunti mostrano valori migliori. Infine, la misura 4 mostra una lieve sovrapposizione tra le distribuzioni delle accuracy. I risultati sono tendenzialmente migliori con i riassunti.

Si riportano ora i boxplot relativi alle diverse accuracy ottenute con il modello DistilBERT.




DistilBERT
Accuracy misura 2
0.6
0.5
0.4
0.3
Testi originali
Riassunti PEGASUS
Riassunti BART


DistilBERT
Accuracy misura 3
0.50
0.45
0.40
0.35
0.30
0.25
Testi originali
Riassunti PEGASUS
Riassunti BART


DistilBERT
Accuracy misura 4
0.50
0.48
0.46
0.44
0.42
0.40
0.38
Testi originali
Riassunti PEGASUS
Riassunti BART

Le considerazioni sono simili a quelle fatte per il Doc2vec+SVM, con la differenza che le distribuzioni per la misura 1 sono ancora più sovrapposte.

In generale si può affermare che l'approccio alla classificazione basato sulla summarization può migliorare le performance rispetto alla classificazione sui testi originali. I miglioramenti sono più apprezzabili nelle metriche che valutano il comportamento del classificatore sulle categorie vere. La complessità dei modelli Transformer non ha però consentito di migliorare i risultati del Doc2vec. Questo può essere dovuto a una quantità di dati di training non sufficiente per sfruttare al meglio la complessità di queste architetture.

Capitolo 6

Conclusioni

Dai risultati del lavoro svolto è emerso come le tecnologie in ambito NLP possono essere interessanti strumenti nel contesto di processi aziendali. I risultati ottenuti non sono però ancora sufficienti per un effettiva messa in produzione di questi sistemi. Lo scopo di questo capitolo è di illustrare le limitazioni del lavoro svolto e in che modo esso può essere migliorato e ampliato.

6.1 Criticità e sviluppi futuri

Selezione dataset

Un prima considerazione riguarda la selezione del dataset utilizzato. Come illustrato al Capitolo 3, è stata infatti considerata solamente una porzione del dataset originale. La scelta deriva dalla praticità di costruzione dei riassunti di riferimento (4.1.1) a partire dagli attributi presenti in database. Un limite del classificatore costruito è perciò di poter essere applicato solamente ai riassunti di fondi con asset class *Azionari* o *Obbligazionari*, mentre può essere invece applicato ai testi di tutti fondi, non potendo però beneficiare delle migliori performance offerte dai riassunti.

Per generalizzare la costruzione dei riassunti di riferimento è sufficiente considerare i pattern che caratterizzano gli attributi dei fondi delle altre asset class per effettuarne la normalizzazione, come avviene in 3.3.1. Il motivo per cui questa generalizzazione non è stata considerata nel lavoro di tesi è che le relazioni tra i vari attributi sono meno banali rispetto a quelle relative a Azionari e Obbligazionari. Per effettuare questa generalizzazione è quindi necessaria una analisi accurata degli attributi dei fondi.

I fondi del dataset originale vengono inoltre selezionati per comparto. Non è possibile eseguire questa selezione in modo più fine nel contesto del classificatore, dal momento che si avrebbero testi identici per item diversi. Le stesse considerazioni valgono anche per quanto riguarda l'esclusione delle categorie hedged. Per considerare queste ultime è necessario utilizzare come input i dati relativi ai rendimenti dei fondi, che possono consentire di distinguerle dalle rispettive categorie non hedged. Anche con questo tipo di input i fondi del dataset originale appartenenti allo stesso comparto sono comunque indistinguibili, se

non viene prima effettuata una selezione. Questa caratteristica costituisce perciò un limite strutturale sul livello di dettaglio che può essere considerato da un classificatore.

Input alternativi

Per aumentare il livello di dettaglio del classificatore o migliorarne le performance si potrebbero considerare altri tipi di input oltre al campo testuale sulla descrizione del fondo.

Un altro input testuale che può essere utilizzato è il nome del fondo. Esso contiene infatti informazioni molto condensate sul fondo in questione, anche se non ne racchiude tutte le caratteristiche. Questo attributo può essere utilizzato insieme al campo testuale utilizzato in questo lavoro (politica e finalità dell'investimento). Come anticipato precedentemente, un altro possibile input è dato dalle serie storiche dei rendimenti dei fondi. Esso potrebbe infatti, congiuntamente agli input testuali, contribuire a catturare informazioni aggiuntive e migliorare le metriche di classificazione.

In generale, tutti gli altri eventuali input alternativi devono essere estraibili senza sforzo dalle fonti informative sui fondi. Considerare infatti input che richiedano un'elaborazione umana risulterebbe poco utile, dal momento che il classificatore ho come obiettivo l'automatizzazione del processo e lo sforzo umano richiesto dovrebbe, al contrario, essere il minore possibile.

Gestione categoria di chiusura

Un'altra problematica riguarda la gestione della categoria di chiusura *non categorizzati*. Essa infatti impatta pesantemente sulle performance del classificatore sia per la sua mancanza di omogeneità, sia per la sua preponderanza rispetto al resto, in termini di dimensioni.

Un possibile modo di gestire la questione è costruire un classificatore a due fasi. Nella prima fase esso dovrebbe essere addestrato su una classificazione binaria per riuscire a distinguere l'appartenenza o meno alla categoria di chiusura. Successivamente, esso può essere costruito per effettuare la classificazione sulle sole categorie vere. In questo modo, differenziando le due operazioni, si avrebbe a che fare con classi meno sbilanciate in entrambe le fasi.

Si può però osservare che, anche operando nel modo appena descritto, il dataset resterebbe comunque sbilanciato, come si può notare alla figura 3.6. Per gestire lo sbilanciamento del dataset si può ricorrere a particolati tecniche di campionamento, come l'oversampling [5] o l'undersampling. Anche con questa soluzione si può valutare se applicare queste tecniche direttamente su unico classificatore, oppure se separare le due fasi di classificazione e applicarle nella seconda fase.

Diversi algoritmi di classificazione

Per la classificazione nel caso di architettura Doc2vec è stata utilizzata la Support Vector Machine. Per ampliare il panorama dei risultati ottenuti, si possono considerare altri algoritmi di apprendimento statistico, come Random Forest o Naive Bayes Classifier. Si potrebbero anche applicare, come nel caso di architetture Transformer, reti neurali per la

classificazione a partire dagli embedding Doc2vec. Allo stesso modo, per quanto riguarda la classificazione basata su architetture Transformer, si può considerare la possibilità di applicare algoritmi di apprendimento statistico sugli embedding prodotti, per mettere i risultati a confronto con quelli generati da reti neurali.

6.2 Considerazioni finali

Nel lavoro di tesi, i risultati ottenuti nella generazione automatica di riassunti per catturare le informazioni principali sono sicuramente positivi. Oltre a generare un un testo coerente e qualitativamente allineato ai contenuti da rappresentare, i modelli di summarization si sono dimostrati utili anche ai fini della classificazione. Le metriche di classificazione hanno mostrato infatti miglioramenti con l'utilizzo dei testi riassunti rispetto ai testi originali. Le architetture Transformer sono strumenti fondamentali per l'obiettivo di text summarization e sono sicuramente la principale tecnologia da tenere in considerazione nel constesto di task generativi. Per quanto riguarda invece la classificazione testuale, oltre ai Transformer, non bisogna trascurare l'utilizzo di tecnologie più vecchie, come Doc2vec. Esso ha infatti generalmente mostrato risultati leggermente migliori rispetto ai Transformer, nonostante la sua semplicità architetturale.

In conclusione, bisogna sottolineare come sia emersa la notevole complessità nel cercare di automatizzare i processi di classificazione in ambito finanziario. La grande varietà e disomogeneità dei fondi comuni d'investimento è ciò che rende questo obiettivo particolarmente complicato da raggiungere pienamente. Attraverso lo sviluppo e l'integrazione dei diversi metodi di classificazione si spera però di poter ottenere risultati sempre migliori.

Appendice A

Reti neurali

Le reti neurali artificiali sono una tipologia di modello di apprendimento automatico di cruciale importanza in tutti i contesti in cui si devono catturare relazioni complesse. In particolare, esse sono largamente utilizzate nell'ambito del Natural Language Processing. Lo scopo di questa appendice è descriverne le principali caratteristiche funzionali al lavoro svolto, facendo principalmente riferimento a [26].

Le reti neurali sono rappresentabili come dei grafi, cioè oggetti caratterizzati da nodi collegati da archi (detti *connessioni*). I nodi sono disposti in fila su diversi strati (*layer*). Gli archi collegano i nodi tra layer successivi; se ogni nodo di un layer è collegato a tutti i nodi del layer successivo e di quello precedente, il layer è detto *completamente connesso*. A ogni connessione è associato un peso; per ogni un nodo viene calcolata la somma dei valori dei nodi precedenti che sono collegati a esso, pesata per i relativi valori degli archi. Può essere inoltre sommato un ulteriore termine detto *bias*. Il valore ottenuto viene poi dato in input a una particolare funzione detta *funzione di attivazione* il cui output costituisce il valore assunto dal nodo di arrivo.

L'addestramento di una rete neurale dipende dalla sua architettura, ma l'obiettivo è, in generale, di ottimizzare il valore dei pesi delle connessioni sulla base di una funzione che dipende da essi.

Reti Feed-Forward

Le reti neurali con la architettura più semplice da illustrare sono le reti di tipo *feed-forward*. Esse cono costituite da:

- un layer di input, i cui valori dei nodi sono dati dalle componenti del vettore dato in input alla rete;
- uno o più layer nascosti, completamente connessi tra loro e con i layer di input e di output;
- un layer di output che fornisce il valore o il vettore predetto dalla rete.

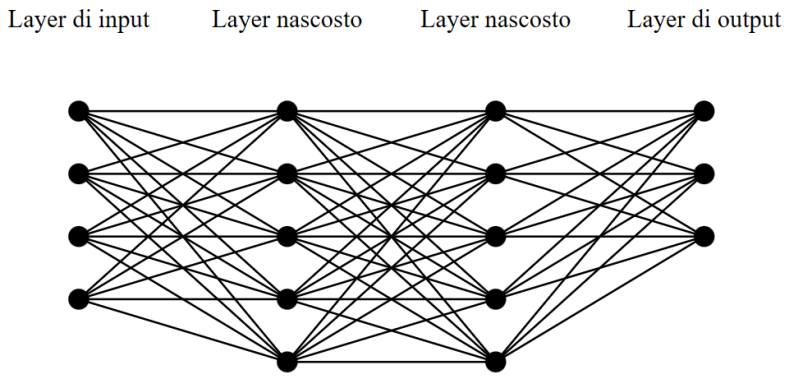


Figura A.1. Rappresentazione di una rete neurale feed-forward.

Funzioni di attivazione

I valori dei nodi non vengono semplicemente combinati con trasformazioni affini sulla base dei pesi delle connessioni dei bias, ma vengono dati in input alla funzione di attivazione scelta. Questa componente è fondamentale per introdurre non linearità nel modello. Infatti, se non ci fossero funzioni di attivazione, il susseguirsi di combinazioni affini a partire dal vettore di input potrebbe essere accorpato in un'unica trasformazione affine.

Vi sono diverse tipologie di funzione di attivazione, si riportano nel seguito le più frequenti.

- la Rectified Linear Unit (detta ReLU) è una funzione che vale 0 fino a una certa soglia, dopo la quale diventa lineare. Ha lo scopo di attivare il neurone solamente se l'input ricevuto supera la soglia.

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{altrimenti} \end{cases}$$

- La Gaussian Error Linear Unit (GELU), presentata in [13], è un'evoluzione della ReLU basata sulla distribuzione normale. Essa, oltre ad essere differenziabile in ogni punto, ha mostrato performance migliori ed è per questo largamente utilizzata nelle recendi architetture di reti neurali. La funzione GELU può essere approssimata dalla seguente formula [13]:

$$\mathrm{GELU}(x) = 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

- La funzione sigmoide è una funzione a una variabile che assume valori tra 0 e 1. È comunemente usata come funzione di attivazione nel layer di output per ottenere un

valore di probabilità nel caso di classificazione binaria.

$$\text{Sigmoide}(x) = \frac{e^x}{1+e^x}$$

- La funzione softmax può essere vista come una generalizzazione n-dimensionale della precedente, andando a mappare un vettore di dimensione n in un vettore della stessa dimensione con supporto $[0,1]^n$. È utilizzata per la classificazione multiclasse.

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, ..., n$$

Funzioni di loss

L'addestramento di una rete neurale avviene attraverso la minimizzazione di una funzione obiettivo, detta *funzione di loss*. Tale funzione ha l'obiettivo di misurare di quanto le previsioni fatte dalla rete si discostano dai valori reali. A seconda del tipo di algoritmo possono essere utilizzate diverse funzioni di loss. Si elencano le loss per la classificazione più rilevanti nel lavoro svolto. Con la notazione utilizzata, in riferimento a [26], si indica con W le matrici dei pesi e con b i vettori dei bias. Con $\hat{y}_{ij}$ ci si riferisce invece all'output per la classe j del il campione i-esimo.

- Hinge loss. Essa è spesso utilizzata in classificatori binari per cui i dati possono essere categorizzati in $j \in \{-1,1\}$.

$$L(W,b) = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y_{ij} \cdot \hat{y}_{ij}\right)$$

  Dato che l'output è binario, j può essere scelto pari a -1 o a 1 in modo indifferente.

- Cross-entropy loss. Essa è anche detta log verosimiglianza negativa e misura la somiglianza tra due distribuzioni di probabilità. Il segno meno serve a renderla una funzione da minimizzare.

$$L(W,b) = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log\left(\hat{y}_{ij}\right)$$

Backpropagation

L'addestramento di una rete neurale avviene attraverso l'ottimizzazione di tutti i parametri (pesi e bias relativi alle connessioni) in modo da minimizzare la funzione di loss. Per ottimizzare i parametri è fondamentale poter calcolare il gradiente della funzione rispetto a essi. Questa operazione viene eseguita attraverso un algoritmo detto *backpropagation*, la cui descrizione è tratta da [1].

La fase preliminare dell'algoritmo, detta *forward phase* consiste nel dare in input i dati di addestramento per ottenere il valore di output della funzione di loss sulla base dei pesi iniziali. Viene poi calcolata la derivata della funzione di loss rispetto all'output:

$$\frac{\partial L}{\partial o}.$$

Nella fase successiva, detta *backward phase*, viene applicata la regola della catena per ottenere le derivate parziali rispetto a tutti i parametri della rete. Se si indica con $w(h_{r-1}, h_r)$ il peso relativo alla connessione tra il neurone h_{r-1} è il neurone h_r, il valore della derivata parziale della loss rispetto a esso è dato da:

$$\frac{\partial L}{\partial w(h_{r-1}, h_r)} = \frac{\partial L}{\partial o} \cdot \left[\sum_{[h_r, \dots h_k, o] \in \mathcal{P}} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \cdot \frac{\partial h_r}{\partial w(h_{r-1}, h_r)},$$

dove con $h_r, \dots h_k, o$ si indica una possibile sequenza di neuroni da h_r all'output e con $\mathcal{P}$ si indicano tutti i percorsi esistenti. Il problema nel calcolo dell'espressione precedente è che la sommatoria su $[h_r, \dots h_k, o] \in \mathcal{P}$ ha un numero di termini che aumenta esponenzialmente al crescere di k. Per risolvere questo problema computazionale viene applicata una strategia basata sulla programmazione dinamica, descritta in [1]. Calcolando tutte le derivate parziali si ottiene il valore del gradiente rispetto a tutti i parametri della rete.

Ottimizzazione e discesa del gradiente

Il calcolo del gradiente serve a minimizzare il valore della funzione di loss con algoritmi di ottimizzazione basati sulla *discesa del gradiente*. La strategia classica consiste nel calcolare il gradiente sulla base dei dati di addestramento, aggiornare i parametri eseguendo uno step nel verso opposto a quello del gradiente (dato che è una direzione di discesa) e iterare il processo fino ad arrivare alla convergenza. La lunghezza dello step è denotata con α ed è detto *learning rate*. Se si indica con θ l'insieme di tutti i parametri del modello, il loro aggiornamento avverrà iterativamente nel seguente modo:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla L(\theta_t).$$

Questo approccio è però di difficile implementazione nel caso di reti molto complesse con una grande mole di dati di addestramento. Infatti, una singola iterazione per eseguire uno step di discesa del gradiente richiederebbe uno sforzo computazionale troppo elevato. Per questo motivo viene utilizzata una procedura detta *discesa stocastica del gradiente*. La modalità con cui questo approccio viene più frequentemente implementato è mediante l'utilizzo dei cosiddetti *mini-batch*. I dati di training vengono suddivisi in gruppi (batch) ognuno dei quali viene utilizzato per effettuare il calcolo del gradiente ed eseguire uno step. In questo modo ogni batch approssima il valore del gradiente che si otterrebbe con tutti i dati, riducendo però lo sforzo computazionale. L'addestramento mediante tutti i batch, cioè dopo l'intero utilizzo dei dati di training è detto *epoch*.

In alcuni casi, è necessario utilizzare batch molto piccoli per rendere la rendere la rete computazionalmente sostenibile. Una possibile conseguenza negativa di questa scelta è uno scarsa convergenza dell'algoritmo, dal momento che batch molto piccoli (constituiti anche da un solo elemento) possono generare un gradiente troppo specifico sul singolo batch e non abbastanza rappresentativo di tutti i dati di training. Questo problema viene risolto con la tecnica dell'*accumulazione del gradiente*. Essa consiste nel calcolare le loss sui singoli batch e accorparne un certo numero (attraverso una media) per calcolare un unico gradiente sul quale eseguire uno step di discesa. Così facendo, il gradiente sarà più

rappresentativo dei dati, ma il suo calcolo risulterà più agevole rispetto al calcolo diretto su batch troppo grandi.

Adam e AdamW

Una delle questioni più importanti nell'algoritmo di discesa del gradiente è la scelta della lunghezza del passo nella direzione di discesa. Sono state sviluppate infatti diverse evoluzioni degli algoritmi di discesa del gradiente, in particolare per il calcolo della lunghezza dello step di discesa. Esso può essere calcolato sulla base di strategie adattive, cioè in modo da dipendere dal valore assunto dal gradiente. Una delle tecniche attualmente più diffuse e utilizzate nei modelli Transformer è l'ottimizzatore *Adam*, presentato nel 2014 in [15]. Esso modifica l'ampiezza dello step di discesa calcolando le seguenti quantità.

$$
\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla L(\theta_t) \\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot [\nabla L(\theta_t)]^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t}
\end{aligned}
$$

β_1 e β_2 sono detti *tassi di decadimento esponenziale* mentre con β_1^t, β_2^t si indicano le potenze t-esime. Nella discesa del gradiente, ogni step viene eseguito nel seguente modo:

$$
\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},
$$

dove il parametro ϵ assume valori piccoli, come 10^{-8} [15].

Sono stati proposti diversi algoritmi per migliorare le performance di Adam, tra cui Adafactor [34] (utilizzato nel pre-training di alcune architetture Transformer) e AdamW [22], utilizzato come ottimizzatore di default nella libreria Transformers. Si riporta una descrizione di quest'ultimo, in riferimento al paper [22].

AdamW è simile all'agoritmo Adam, ma presenta l'utilizzo di un nuovo parametro, cioè il *weight decay*, coefficiente utilizzato per introdurre bias, ripreso da [11]. Nel paper di AdamW, l'algoritmo presentato include anche l'utilizzo degli *schedule multiplier*, sequenze di elementi che possono modulare il passo di discesa a seconda del numero di iterazioni. In AdamW, il calcolo di m_t, v_t, $\hat{m}_t$, $\hat{v}_t$ viene eseguito in modo uguale ad Adam. Successivamente lo step di discesa avviene nel seguente modo:

$$
\theta_t = \theta_{t-1} - \eta_t \left(\frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)
$$

ν_t è lo schedule multiplier relativo alla t-esima iterazione e λ è il weight decay. Nella libreria Transformers gli schedule multiplier di default sono di tipo *linear* con una fase di *warmup*. Questo significa che nella fase di warmup essi crescono linearmente da 0 a 1, e successivamente decrescono linearmente da 1 a 0 fino alla fine del training. Gli altri parametri di default sono $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ e $\lambda = 0$.

Bibliografia

[1] C. C. Aggarwal. *Neural networks and deep learning: a textbook*. Springer, 2018.

[2] H. K. Baker, G. Filbeck, and Kiymaz H. *Mutual Funds and Exchange-Traded Funds: Building Blocks to Wealth*. Oxford University Press, 2015.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. *A Training Algorithm for Optimal Margin Classifiers*. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, page 144–152. Association for Computing Machinery, 1992.

[4] P. Brandimarte. *An Introduction to Financial Markets: A Quantitative Approach*. John Wiley & Sons Inc, 2018.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. *SMOTE: Synthetic Minority Over-sampling Technique*. DOI: `https://doi.org/10.1613/jair.953`, 2011.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. DOI: `https://doi.org/10.48550/arXiv.1810.04805`, 2019.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. `https://github.com/google-research/bert/blob/master/tokenization.py`, 2020.

[8] J. Eisenstein. *Introduction to Natural Language Processing*. The MIT Press, 2019.

[9] FIDA Finanza Dati Analisi. *Sistema FIDARating*. 2018.

[10] P. Gage. *A new algorithm for data compression*. *The C Users Journal archive*, 12:23–38, 1994.

[11] S. J. Hanson and L. Y. Pratt. *Comparing biases for minimal network construction with back-propagation*. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, page 177–185, 1988.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2008.

[13] D. Hendrycks and K. Gimpel. *Gaussian Error Linear Units (GELUs)*. DOI: https://doi.org/10.48550/arXiv.1606.08415, 2016.

[14] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. DOI: https://doi.org/10.48550/arXiv.1503.02531, 2015.

[15] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. DOI: https://doi.org/10.48550/arXiv.1412.6980, 2014.

[16] T. Kudo. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. DOI: https://doi.org/10.48550/arXiv.1804.10959, 2018.

[17] H. Lane, C. Howard, and H. Hapke. *Natural Language Processing in Action: Understanding, analyzing, and generating text with Python*. Computer Science Department. Paper 1529, 2019.

[18] A. Lenci. *Distributional Models of Word Meaning*. DOI: https://doi.org/10.1146/annurev-linguistics-030514-125254, 2018.

[19] M. Lewis, Y. Liu, N. Goyal, M Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. DOI: https://doi.org/10.48550/arXiv.1910.13461, 2019.

[20] C. Lin. *ROUGE: A Package for Automatic Evaluation of Summaries*. In *Text Summarization Branches Out*, pages 74–81. Association for Computational Linguistics, 2004.

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. DOI: https://doi.org/10.48550/arXiv.1907.11692, 2019.

[22] I. Loshchilov and F Hutter. *Decoupled Weight Decay Regularization*. DOI: https://doi.org/10.48550/arXiv.1711.05101, 2019.

[23] H. Markowitz. *Portfolio Selection*. In *Journal of Finance*, volume 7(1), pages 77–91. American Finance Association, 1952.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. DOI: https://doi.org/10.48550/arXiv.1301.3781, 2013.

[25] T. Mikolov and Q. Le. *Distributed Representations of Sentences and Documents*. DOI: https://doi.org/10.48550/arXiv.1405.4053, 2014.

[26] J. Patterson and A. Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly, 2017.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. *Language Models are Unsupervised Multitask Learners.* `https://api.semanticscholar.org/CorpusID:160025533`, 2019.

[29] R. Řehůřek and P. Sojka. *Software Framework for Topic Modelling with Large Corpora.* In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[30] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* DOI: `https://doi.org/10.48550/arXiv.1910.01108`, 2019.

[31] M. Schuster and K. Nakajima. *Japanese and Korean voice search.* In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[32] A. See, P. J. Liu, and C. D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks.* In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083. Association for Computational Linguistics, 2017.

[33] R. Sennrich, B. Haddow, and A. Birch. *Neural Machine Translation of Rare Words with Subword Units.* DOI: `https://doi.org/10.48550/arXiv.1508.07909`, 2016.

[34] N. Shazeer and M. Stern. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost.* DOI: `https://doi.org/10.48550/arXiv.1804.04235`, 2018.

[35] A. Sherstinsky. *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network.* DOI: `https://doi.org/10.1016%2Fj.physd.2019.132306`, 2020.

[36] L. Tunstall, L. von Werra, and T. Wolf. *Natural Language Processing with Transformers.* O'Reilly, 2022.

[37] Carnegie-Mellon University. *Speech understanding systems: summary of results of the five-year research effort at Carnegie-Mellon University.* Computer Science Department. Paper 1529, 1977.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need.* DOI: `https://doi.org/10.48550/arXiv.1706.03762`, 2017.

[39] A. Viterbi. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[40] K. Whistler. *Unicode Normalization Forms*. `https://www.unicode.org/reports/tr15/tr15-54.html`, 2023.

[41] T. Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. DOI: `https://doi.org/10.48550/arXiv.1910.03771`, 2019.

[42] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. DOI: `https://doi.org/10.48550/arXiv.1508.07909`, 2016.

[43] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. DOI: `https://doi.org/10.48550/arXiv.1912.08777`, 2020.

[44] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. *BERTScore: Evaluating Text Generation with BERT*. DOI: `https://doi.org/10.48550/arXiv.1904.09675`, 2020.

Ringraziamenti

Vorrei porgere un particolare ringraziamento al dott. Luca Lodi e a tutti i colleghi che mi hanno assistito nello sviluppo del progetto in azienda. Grazie anche al prof. Paolo Garza e al dott. Moreno La Quatra, per avermi aiutato nella scoperta di una materia per me nuova.

Un grande grazie va alla mia famiglia per avermi sempre sostenuto in questi anni di studi. Infine, grazie a Carola, per aver creduto in me in ogni momento, illuminando tutte le mie giornate.