ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI METODI DI SOFT LABELING PER LA MULTI-DOCUMENT SUMMARIZATION IN AMBITO LEGALE

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Luca Ragazzi

Presentata da
Luca Rubboli

Seconda Sessione di Laurea
Anno Accademico 2021 – 2022

PAROLE CHIAVE

Natural Language Processing

Soft Labeling

Multi-Document Summarization

Dominio Legale

Transformers

A chiunque mi sia stato vicino,
e mi abbia aiutato a raggiungere questo traguardo.

Abstract

In questo elaborato viene trattata l'analisi del problema di soft labeling applicato alla *multi-document summarization*, in particolare vengono testate varie tecniche per estrarre frasi rilevanti dai documenti presi in dettaglio, al fine di fornire al modello di summarization quelle di maggior rilievo e più informative per il riassunto da generare. Questo problema nasce per far fronte ai limiti che presentano i modelli di summarization attualmente a disposizione, che possono processare un numero limitato di frasi; sorge quindi la necessità di filtrare le informazioni più rilevanti quando il lavoro si applica a documenti lunghi. Al fine di scandire la metrica di importanza, vengono presi come riferimento metodi sintattici, semantici e basati su rappresentazione a grafi AMR. Il dataset preso come riferimento è Multi-LexSum, che include tre granularità di summarization di testi legali. L'analisi in questione si compone quindi della fase di estrazione delle frasi dai documenti, della misurazione delle metriche stabilite e del passaggio al modello stato dell'arte PRIMERA per l'elaborazione del riassunto. Il testo ottenuto viene poi confrontato con il riassunto target già fornito, considerato come ottimale; lavorando in queste condizioni l'obiettivo è di definire soglie ottimali di *upper-bound* per l'accuratezza delle metriche, che potrebbero ampliare il lavoro ad analisi più dettagliate qualora queste superino lo stato dell'arte attuale.

Indice

Elenco delle figure

Capitolo 1

Intelligenza Artificiale

L'intelligenza artificiale (AI) è la disciplina che studia i meccanismi alla base dell'apprendimento umano, con la finalità di progettazione di sistemi hardware e software in grado di emularli.

1.1 Introduzione all'Intelligenza Artificiale

Le basi dell'intelligenza artificiale possono essere ricondotte agli anni '50 ma, complici aspettative fin troppo ottimistiche, supporto hardware non all'altezza e mancanza di dati da analizzare, i primi concreti sviluppi si registrano alla fine degli anni '80. È però il recente avvento dell'era dei Big Data e dell'Internet of Things (IoT) che ha incentivato in maniera consistente la ricerca a sviluppare nuove tecnologie nell'ambito di sistemi intelligenti; il bisogno infatti di costante elaborazione di una mole di dati come quella attualmente generata dalla società risulterebbe incolmabile senza l'impiego di modelli avanzati come quelli offerti da questa disciplina.

Ciò che distingue l'approccio classico dei sistemi di elaborazione dei dati da quelli che costituiscono le basi dei modelli intelligenti è l'estrazione della conoscenza dalle osservazioni raccolte; l'accuratezza di questa deriva sia dalla quantità di informazioni disponibili, sia dalla bontà dei dati nel modellare nel modo più preciso possibile il problema.

L'evoluzione dei modelli in questo settore ha portato allo sviluppo di tecniche sempre più sofisticate, che si pongono l'obiettivo di apprendere e migliorare le proprie prestazioni nel tempo, acquisendo un'esperienza derivante dai dati processati, note come `Machine Learning`. Questa branca dell'intelligenza artificiale ha giovato diverse aree di lavoro, tra cui tecniche innovative di recommendation e guida assistita o, nei casi più specializzati, autonoma.

Tuttavia, queste tecniche non consentono di raggiungere risultati particolarmente soddisfacenti in diversi ambiti, tra cui l'elaborazione delle immagini,

per cui è necessario dispiegare sistemi supportati da reti neurali, frutto della ricerca nell'ambito `Deep Learning`. Quest'area si distingue per l'abilità dei sistemi di individuare in maniera autonoma le caratteristiche di maggior spicco dei dati in esame, al fine di migliorare l'effetto dell'elaborazione.

1.2 Modelli di Machine Learning

La disciplina del machine learning si interpone di estrarre modelli di conoscenza da insiemi di dati, detti dataset, al fine di generare previsioni accurate. Questi modelli sono costituiti da parametri, in numero proporzionale alla difficoltà del problema e all'accuratezza che si vuole raggiungere. Alcuni parametri necessitano di adattamento al problema, pertanto viene generalmente adottata una fase di addestramento del modello, mentre altri, definiti *iperparametri*, vengono definiti dal programmatore in fase di creazione.

Al fine di affinare questa fase, viene spesso adottato il metodo di suddivisione dell'insieme dei dati forniti in 3 sottoinsiemi: il *training set*, che prende parte al calcolo e perfezionamento dei parametri, il *validation set*, che stima la generalizzazione del modello e il *test set*, che permette una valutazione realistica del modello su dati ignoti. Secondo il metodo *hold-out* infatti, il validation set permette di stabilire se un modello risulta poco performante sia per problemi di *underfitting*, ossia scarsi risultati in termini di accuratezza sia sul training set sia sul validation set, oppure in termini di *overfitting*, per cui i buoni risultati ottenuti sul training set non si rilevano anche sul validation set. Le cause imputabili al primo caso sono o l'impiego di modelli troppo semplici per descrivere il problema, oppure l'utilizzo di esempi che non descrivono correttamente il problema, mentre il secondo caso denota una scarsa generalizzazione su nuovi dati dovuta o a una complessità del modello fin troppo elevata, con eccessiva aderenza ai dati di training, oppure a uno scarso quantitativo di dati con cui conseguire l'addestramento.

Per ovviare quindi a queste problematiche è necessario:

- Applicare un filtraggio ai dati, avendo premura di rimuovere eventuali caratteristiche non rilevanti;
- Normalizzare le peculiarità dei dati su una scala uniforme, per rendere più stabile la fase di addestramento;
- Regolarizzare i parametri vincolandoli a valori più piccoli, al fine di semplificare e generalizzare il modello;
- Effettuare il *tuning* degli iperparametri mediante tecniche di cross validation, al fine di ottimizzare la struttura del modello;

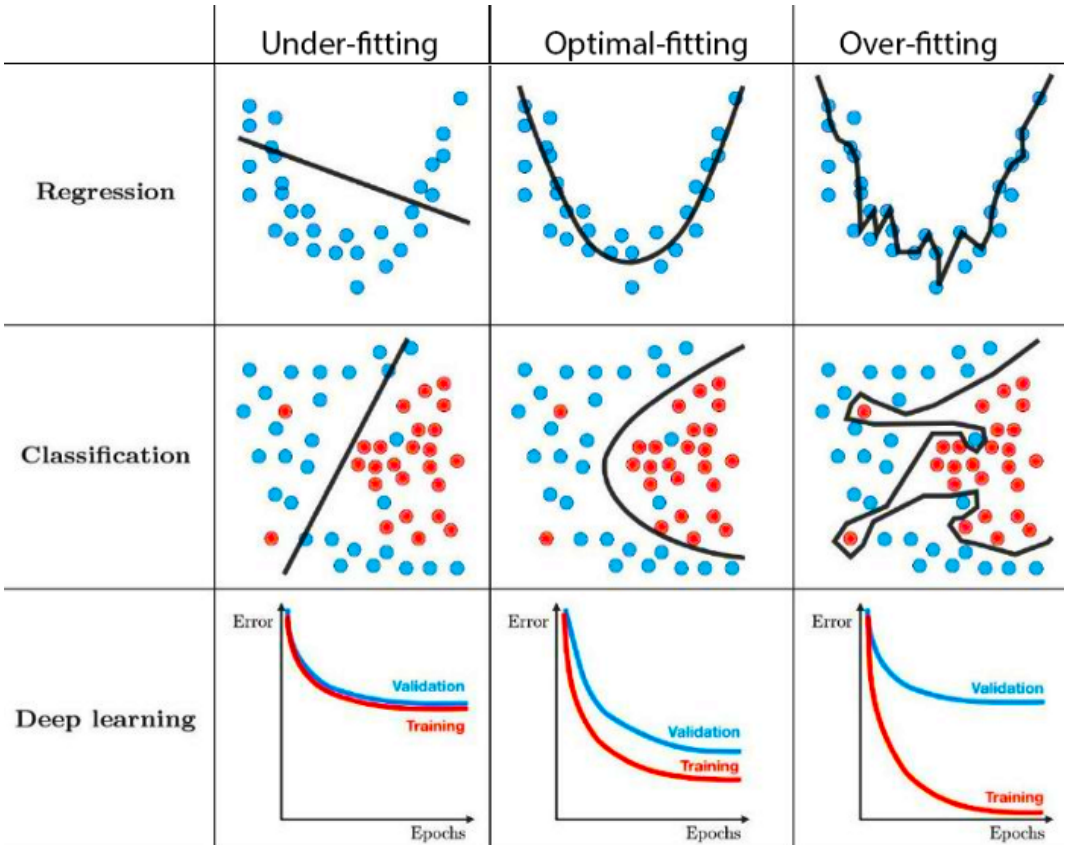

Figura 1.1: Esempi di modelli di varia natura in underfitting, generalizzati e in overfitting.

Fonte: https://towardsdatascience.com

- Aggiungere nuovi dati di training a quelli già utilizzati.

L'obiettivo cardine della fase di addestramento, infatti, è quello di ottenere un modello che minimizza l'errore sui dati modellando in modo generale il problema.

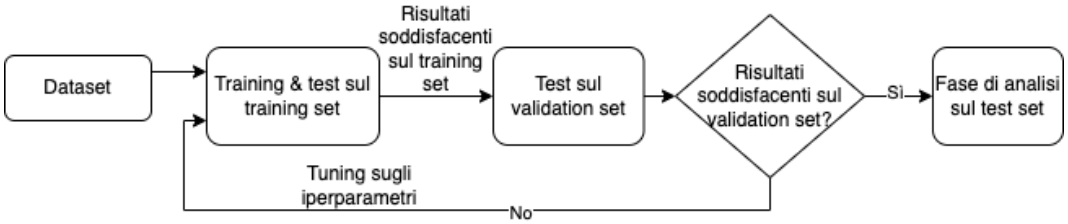

Figura 1.2: Flusso di lavoro nella fase di training e validation.

Dopo aver ottenuto risultati soddisfacenti sia sul training che sul validation set, si passa a una fase di ulteriore verifica con il test set, che sancisce l'effettiva bontà del modello.

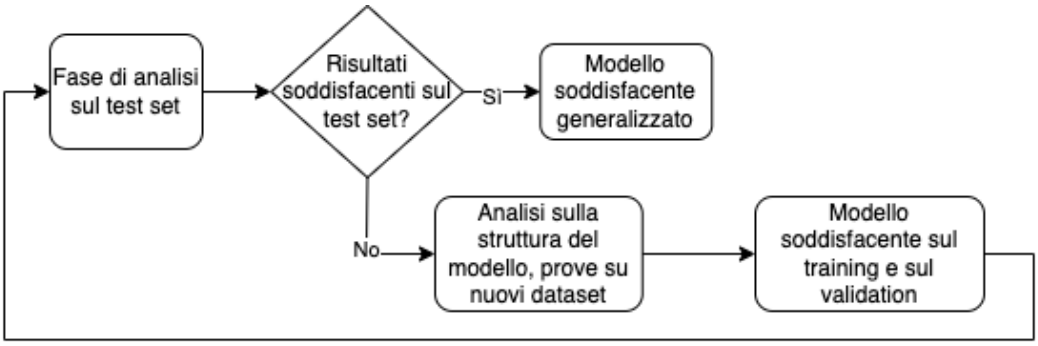

Figura 1.3: Flusso di lavoro nella fase di test.

Si possono distinguere diverse tecniche di addestramento, dipendenti sia dalle caratteristiche dei dati forniti, sia dal modo di estrarre la conoscenza da questi:

- **Apprendimento supervisionato**: qualora il modello abbia a che fare con dataset etichettati, per cui i dati subiscono preliminarmente una fase (spesso onerosa) di *pre-processing*, in cui vengono definite le caratteristiche di spicco e l'*output* che il modello è chiamato a predire. Viene generalmente adottato per problemi di **classificazione**, in cui i dati vengono associati ad una specifica categoria, oppure per problemi di **regressione**, per cui si richiede di predire un valore numerico continuo;

- **Apprendimento non supervisionato**: in questo caso il modello ha a che fare con dati non etichettati, e deve essere in grado di estrapolare in maniera autonoma le caratteristiche correlate dei dati forniti al fine di individuare processi e schemi potenzialmente complessi. Viene impiegato per la risoluzione di problemi di **clustering**, che prevede una classificazione di dati in sottoinsiemi senza esprimere le possibili categorie da utilizzare;

- **Apprendimento per rinforzo**: il modello prende il nome di `agente`, che si deve adattare ad un ambiente e alle iterazioni, che generalmente portano a ricompense, con questo. È con questo sistema di feedback che il sistema acquisisce informazioni sulla bontà del suo comportamento. Viene adoperato, ad esempio, per migliorare le abilità di un sistema intelligente in un gioco.

1.3 Deep Learning

Come definito precedentemente, la disciplina del deep learning è una specializzazione dell'intelligenza artificiale che, per mezzo di reti neurali, ha raggiunto risultati importanti in termini di accuratezza rispetto alle architetture precedentemente utilizzate.

1.3.1 Cella neurale

Ispirandosi al concetto di neurone umano, sono stati concepiti modelli matematici complessi che si basano su celle neurali, costituite da uno o più input e un output ottenuto come elaborazione del segnale in ingresso.

Ogni cella contiene dei pesi che vengono applicati ad ogni input. Definendo X il vettore degli input, W il vettore dei pesi associati, y l'output e ϕ la funzione di attivazione applicata, l'elaborazione processata da una cella neurale semplice è:

$$y = \phi(WX^T + b)$$

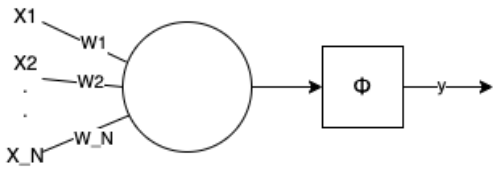

Figura 1.4: Illustrazione della cella neurale.

1.3.2 Layer neurale

Un *layer* neurale è uno strato costituito da più celle neuronali che condividono un impiego comune; generalmente si assegna ad ognuno di questi una funzionalità specifica di riconoscimento di un particolare aspetto del problema.

1.3.3 Rete neurale

Ottenute come concatenazione di più layer neurali collegati a cascata, le reti neurali rappresentano l'architettura moderna più all'avanguardia in ambito intelligenza artificiale. La struttura classica di una rete neurale è composta da un layer di input, uno di output e uno o più strati nascosti (*hidden layer*). Per mezzo di definizione di funzioni di errore o di costi, i layer vengono addestrati a minimizzarle, migliorando vertiginosamente l'accuratezza raggiunta con lo stato dell'arte.

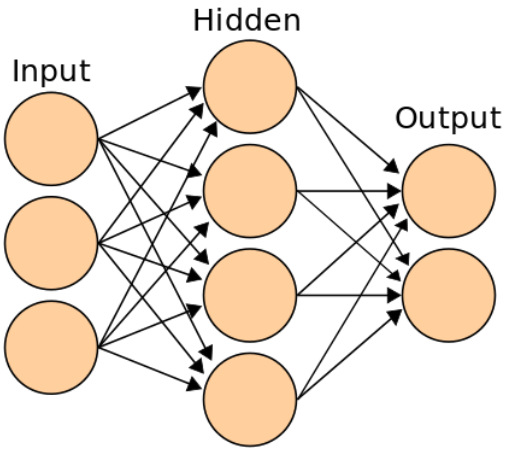

Figura 1.5: Illustrazione di una rete neurale.
Fonte: https://it.wikipedia.org/wiki/Rete$_neurale_artificiale$

1.3.4 Addestramento di reti neurali

Per quanto riguarda l'addestramento supervisionato delle reti neurali, il metodo matematico di riferimento è la `discesa del gradiente`, che prevede di individuare una funzione che esprima l'errore commesso dal modello nell'elaborazione dei dati. Una volta aver trovato questa funzione, partendo da un punto (generalmente casuale), si applica la discesa al fine di raggiungere un minimo locale e di identificare la combinazione di parametri che lo genera.

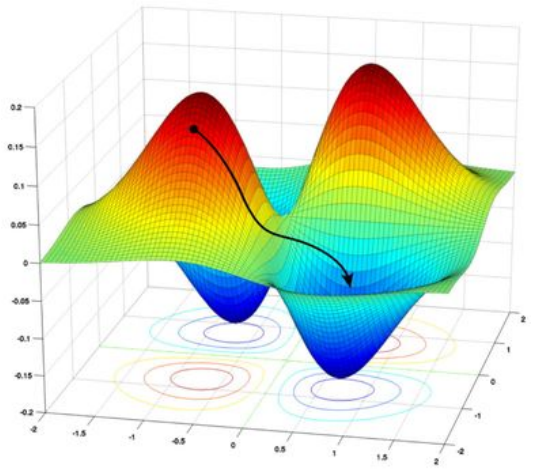

Figura 1.6: Illustrazione della discesa del gradiente.
Fonte: https://www.lescienze.it

Per propagare questa discesa, aggiornando i pesi associati all'input delle celle, viene adottato l'algoritmo di retropropagazione (*backpropagation*) dell'errore.

Questo viene diviso in due passi:

1. *Forward pass*, per cui l'input dato alla rete viene processato e propagato agli strati successivi;

2. *Backward pass*, in cui viene calcolato l'errore commesso sull'elaborazione dell'input e successivamente inviato dall'ultimo strato al primo al fine di aggiornare i pesi associati agli input.

Data la natura di addestramento onerosa in termini di risorse, sono state predisposte soluzioni che tendono ad ottimizzare questa fase, ad esempio la *Batch Normalization*, che prevede di normalizzare l'input di ogni layer, al fine di coordinare l'addestramento dei vari strati ottenendo gradienti più predittivi ed una convergenza più rapida.

Diffuso in molti ambiti, il *Transfer Learning* è un metodo avanzato che prevede l'impiego di modelli generali pre-addestrati in contesti più specifici semplicemente adottando una fase di *fine-tuning*. In questa fase il modello viene addestrato con input mirati e di dimensioni ridotte per specializzarlo in un'area di lavoro. È così possibile evitare o ridurre sensibilmente l'oneroso addestramento delle reti neurali.

Capitolo 2

Natural Language Processing

Il `natural language processing` (NLP) è la disciplina che si occupa di approfondire gli aspetti che riguardano l'elaborazione del linguaggio, espresso in forma testuale o parlata. La natura complessa dell'elaborazione del linguaggio naturale deriva dalla presenza di più lingue, basate su regole grammaticali, vocaboli e talvolta alfabeti differenti. Una stessa lingua può articolarsi in dialetti o forme linguistiche diverse e, a seconda del contesto, assumere significati completamente differenti. Sviluppano un ulteriore livello di ambiguità figure retoriche quali ironia e sarcasmo, oltre che abbreviazioni ed elementi che astraggono dall'idioma stesso, come ad esempio le emoticon.

La forte necessità di contesto del problema rende cruciale l'impiego di tecniche di *pre-processing* per migliorare l'accuratezza dell'elaborazione.

2.1 Background

Sebbene a livello storico siano stati individuati vari tentativi di approccio alla materia, la disciplina nasce negli anni '40 quando, durante la Seconda Guerra Mondiale, è emerso il problema di comunicare rapidamente con altre fazioni e codificare/decodificare messaggi.

La nota pubblicazione di Alan Turing risalente al 1950, "Computing Machinery and Intelligence" [1], stabilisce come criterio di intelligenza di un sistema artificiale l'essere in grado di sostenere una conversazione in maniera indistinguibile da quella sostenuta da un umano.

Le prime tecniche adoperate per problemi di traduzione coinvolgono due tipi di approcci:

- Simbolico, detto *rule-based*, per cui l'elaborazione si basa su complesse regole scritte da linguisti e ricercatori in questo ambito;

- Stocastico, per cui furono privilegiati studi basati su regole statistiche e metodi probabilistici per definire *pattern* del linguaggio naturale.

L'analisi stocastica ha favorito l'introduzione dell'intelligenza artificiale e, attualmente, l'adozione di algoritmi di deep learning ha portato allo stato dell'arte.

2.2 Tecniche di pre-processing

Il testo in linguaggio naturale è un tipo di dato non strutturato; come accennato in precedenza, è di cruciale importanza l'elaborazione preliminare, per normalizzare e contestualizzare il dato grezzo.

Con queste finalità, sono state disposte varie tecniche:

- *Tokenization*, consiste nella divisione di un testo in frasi e, successivamente, in parole. Per un'esecuzione accurata, vengono adottati modelli e regole specifiche della lingua a cui il testo afferisce;

- *Part of Speech* (POS), prevede l'etichettatura, parola per parola, della classe grammaticale di appartenenza;

- *Casefolding*, il cui obiettivo è eliminare ambiguità derivanti da lettere maiuscole piuttosto che minuscole, convertendo il testo in uno standard comune;

- Rimozione delle *stopword*, parole quali articoli, che non danno accesso ad informazioni vere e proprie; generalmente viene stilata una lista di queste preliminarmente;

- Lemmatizzazione, ossia la sostituzione di ogni parola con il suo lemma, la sua forma base;

- *Stemming*, simile alla lemmatizzazione, prevede la sostituzione di ogni parola con la sua radice morfologica, richiede generalmente un algoritmo più semplice, però può portare a perdita di informazione;

- *Word Sense Disambiguation*, è la procedura che ha l'obiettivo di assegnare ad ogni parola il suo significato esatto, identificandola in un preciso contesto; esistono vari algoritmi, alcuni selezionano come definizione corretta quella che contiene il maggior numero di parole vicine a quella presa in dettaglio;

- *Named Entity Recognition*, prevede la ricerca delle *named entity*, specifiche entità a cui si fa riferimento, tra cui luoghi, ruoli lavorativi, ma anche numeri e date; queste sono generalmente considerate come elementi molto rilevanti al fine di contestualizzare una determinata frase.

2.3 Tecniche di rappresentazione del testo

Un altro aspetto di particolare rilievo riguarda la rappresentazione del testo, in quanto i modelli di machine learning non sono in grado di processare i dati in forma testuale. La conversione avviene per mezzo di diverse tecniche il cui obiettivo è di ottenere una rappresentazione compatta e strutturata, tra cui:

- Rappresentazione *Bag of Words*, per cui si stila una tabella composta da due colonne, una relativa ai termini, generalmente standardizzati per mezzo di stemming al fine di ridurre il numero di parole distinte e una relativa al numero di occorrenze nel testo. Questa tecnica risulta particolarmente efficace in materia di *sentiment analysis*, che prevede la classificazione di un testo in categorie; un esempio è quello di classificare una recensione come positiva o negativa, per cui si fa riferimento ad un conteggio di parole positive e negative, assegnando un peso a questi;

- *Vector space model*, offre una rappresentazione di più documenti seguendo una metrica bag of words delle parole di un dizionario comune. Offre un valido contributo all'identificazione degli argomenti trattati nei vari testi; l'integrazione con schemi *term frequency - inverse document frequency* permette di stabilire la rilevanza di un termine in un documento con un fattore term frequency (tf), pari al numero di occorrenze nel testo, e un fattore inverse document frequency (idf), pari al numero di documenti presi in analisi in cui questo occorre, che agisce in maniera inversamente proporzionale: maggiore è il numero di documenti in cui il termine è presente, minore sarà il peso assegnatogli;

- Rappresentazione *n-gram*, raccoglie, ad ogni iterazione, le sequenze di parole di lunghezza n. Queste sequenze possono identificare entità o costruzioni descritte da più parole, andando ad arricchire la struttura bag of words precedentemente descritta. L'architettura a finestra di lunghezza variabile risulta significativa come modello probabilistico in fase di addestramento per la generazione di parole, per cui considerando le $n-1$ parole precedenti, il compito è quello di generare la prossima parola;

2.3.1 Word embedding

La famiglia di tecniche per la rappresentazione del testo sotto forma di vettori numerici che ne preservano la semantica è chiamata word embedding.

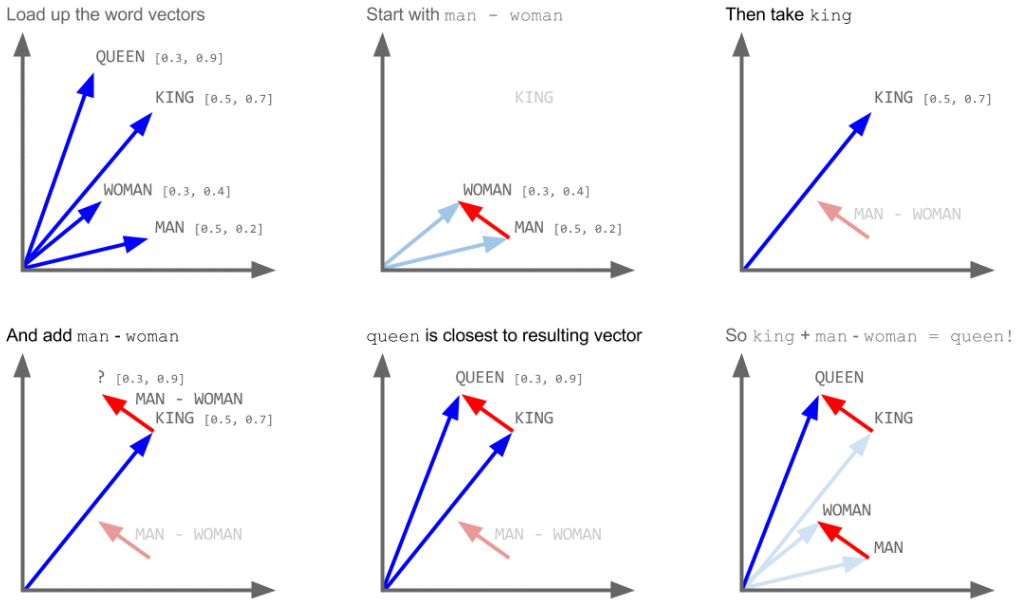

Figura 2.1: Esempio di proiezione di termini in uno spazio vettoriale basato sulla semantica.

Fonte: https://multithreaded.stitchfix.com

Generalmente queste sono di tipo non supervisionato, le etichette per addestrare il modello vengono estratte direttamente dai dati presi in esame. La conversione avviene per mezzo di algoritmi che proiettano le parole in uno spazio vettoriale che mantiene la semantica:

- `One-Hot Encoding` prevede la conversione di ogni parola in un vettore con un solo valore a 1, corrispondente alla posizione occupata nel documento preso in analisi, e tutti gli altri a 0. Questo consente di generare un vettore distinto per ogni parola nel documento; è un algoritmo poco scalabile in termini di lunghezza di documento, perchè porta ad un'esplosione dimensionale all'aumentare delle parole e non mantiene un contesto semantico;

- *Word2Vec* [2] è tra gli algoritmi di embedding più utilizzati, si compone di 2 architetture distinte per l'addestramento, *CBOW*, con il compito di predire una parola dato un contesto, ovvero una serie di parole precedenti

e successive a quella da predire e Skip-gram, con il compito di predire il contesto data una parola.

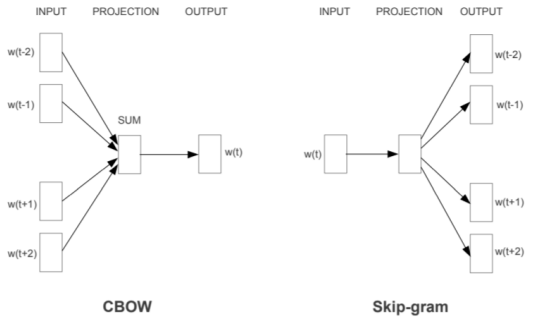

Figura 2.2: Struttura CBOW - Skip gram.
Fonte: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

Maggiore è la mole di dati processata in fase di addestramento, peggiore sarà l'accuratezza nel generare uno spazio vettoriale dove le parole vengono proiettate secondo una metrica di similarità semantica. Esistono altri algoritmi gettonati in quest'ambito, che prevedono la costruzione di matrici di co-relazione tra termini, mediante la quale è possibile definire una rappresentazione accurata delle parole (ad esempio GloVe [3]), o altri che permettono di confrontare le similarità tra documenti traducendoli come vettori numerici (tra cui Doc2Vec [2]).

2.4 Modelli di Deep Learning

L'integrazione di metodologie e sistemi di intelligenza artificiale ha reso possibile un grande miglioramento in termini di accuratezza: le nuove tecnologie in ambito deep learning hanno permesso di ottenere il nuovo stato dell'arte.

2.4.1 Recurrent Neural Network

Le recurrent neural network sono una classe di reti neurali che presenta, oltre alla classica rete *feed-forward* che processa l'input in un'unica direzione, una connessione *back-ward* che permette di processare, al tempo $t+1$, l'input e l'output generato al passo t, introducendo di fatto un concetto di memoria legato alle elaborazioni precedenti. Le celle neurali di questo particolare tipo di rete necessitano di un nuovo insieme di pesi da associare all'input precedentemente generato. Per studiare questa configurazione occorre effettuare un'analisi detta *"unrolling through time"*, che consiste nel considerare diversi passi consecutivi.

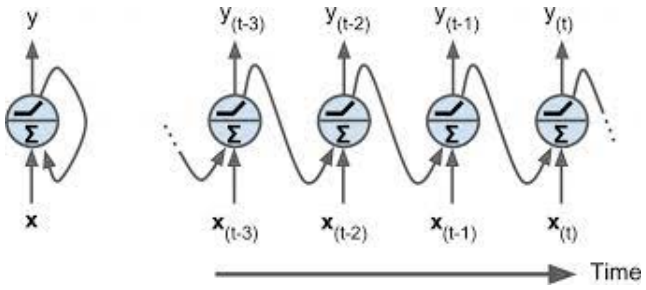

Figura 2.3: Analisi unrolling through time di una RNN.
Fonte: https://andreaprovino.it/rnn-recurrent-neural-network/

Le RNN risultano particolarmente performanti come reti *sequence-to-sequence*, ovvero dove input e output sono costituiti da sequenze di lunghezza arbitraria, *sequence-to-vector* e *vector-to-sequence*, dove uno tra input e output è un elemento unico e non una sequenza. Un limite però dell'architettura è l'addestramento su lunghe sequenze, che tendono a formare gradienti instabili e causano il problema di perdita di memoria, che risulta particolarmente limitata. L'instabilità del gradiente può essere arginata adottando normalizzazioni a livello di *batch* o di layer, che stabilizzano i gradienti agendo sulle feature e sulla loro dimensionalità. Per agire invece sul problema di memoria, sono state predisposte nuove celle neurali con l'integrazione di *gate* che migliorano la gestione della memoria interna.

2.4.2 Long Short-Term Memory cell

Il funzionamento di questa tipologia cella è molto simile a quello di una standard, tuttavia l'addestramento risulta convergere più velocemente e gestisce in maniera migliore la memoria. Lo stato viene diviso in due, uno long-term e uno short-term, mentre i gate aggiuntivi fan si che l'input venga filtrato e che solo le parti più importanti vengano di fatto memorizzate.

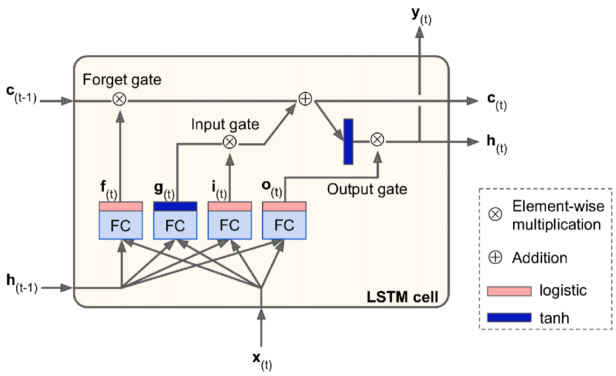

Figura 2.4: Architettura LSTM.
Fonte: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition

2.4.3 Gated Recurrent Unit

Rappresentano una semplificazione della struttura LSTM mantenendone, per alcuni casi, gli stessi benefici. Sfrutta un numero di gate inferiore, ma non sempre porta ad un miglioramento di prestazioni rispetto ad una cella base.

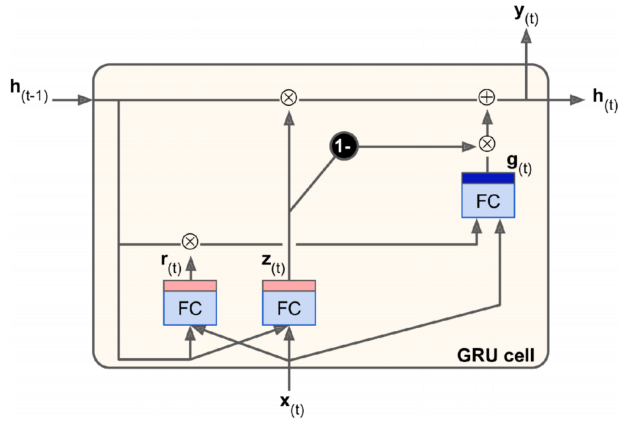

Figura 2.5: Architettura GRU.
Fonte: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition

2.4.4 Convolutional Neural Network

Le convolutional neural network sono una classe di reti neurali la cui struttura connettiva delle celle prende ispirazione dall'organizzazione della corteccia visiva animale. L'architettura si basa su strati convoluzionali che permettono alla rete di estrarre caratteristiche dai dati, applicando dei filtri variabili ad ogni strato. L'addestramento della rete verte sulla specializzazione

dei filtri nell'individuare peculiarità dell'input, pertanto, si prestano al meglio nel campo della computer vision.

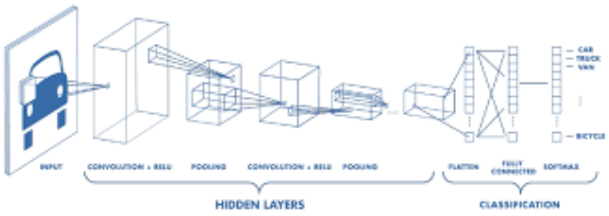

Figura 2.6: Architettura CNN.
Fonte: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

2.4.5 Struttura Encoder-Decoder

Un modello di questa struttura si compone di due parti:

- Encoder: modello sequence-to-vector che prende in input una sequenza ed ignora tutti gli output al di fuori dell'ultimo;
- Decoder: modello vector-to-sequence che prende in input un vettore e restituisce una sequenza in output.

Il modello risultante è di tipo sequence-to-sequence, prende in input una sequenza e ne restituisce una nuova. Si presta in diversi ambiti NLP, tra cui la traduzione di testo per cui, prima di generare l'output, deve essere processata tutta la frase in input.

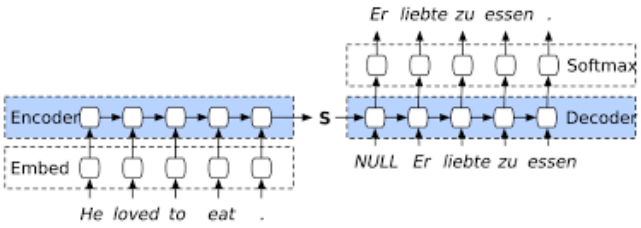

Figura 2.7: Struttura Encoder-Decoder.
Fonte: https://towardsdatascience.com

Attention

L'algoritmo di attention nasce come strumento integrativo alla struttura classica di encoder decoder al fine di migliorare la limitazione dovuta alla scarsa memoria del modello e, conseguentemente, alle limitazioni in termini di

lunghezza dell'input. Questo meccanismo permette al modello di concentrarsi sulle parti di input ritenute più importanti, modificando la struttura di encoder in sequence-to-sequence; l'output atteso dell'encoder è l'elaborazione totale dell'input, e non più solo l'ultima. Il decoder, ad ogni iterazione, sfrutta la sua memoria per computare una somma pesata del contesto passato dall'encoder, che permette di stabilire le parole di input su cui l'elaborazione attuale si deve concentrare. I pesi relativi alla somma vengono generati dall'*attention layer*, che viene addestrato congiuntamente al modello, analizzando quanto l'output che ad ogni passo viene prodotto dal decoder sia allineato con il contesto. Questo meccanismo prende il nome dal suo ideatore, Bahdanau [4].

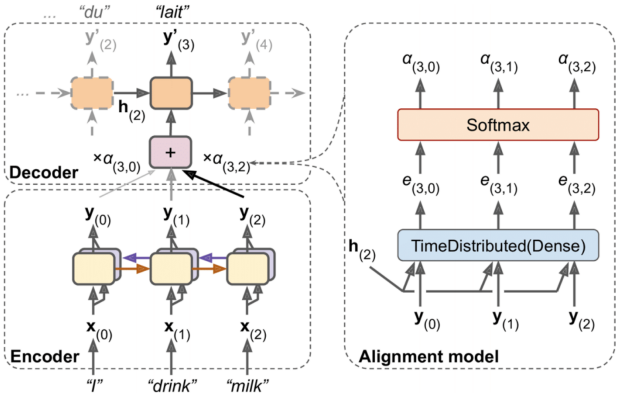

Figura 2.8: Meccanismo di attention integrato a struttura encoder-decoder.
Fonte: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition

2.5 Transformer

Transformer è un'architettura creata dal team di ricerca di Google e rilasciata con l'articolo "Attention is all you need" [5] che ha portato grandi passi avanti per ciò che riguarda lo stato dell'arte di molti problemi di NLP, senza utilizzare strati recurrent e convolutional, ma ottenendo un modello che permette un addestramento più veloce e parallelizzabile, gli elementi della sequenza in input non vengono necessariamente processati in ordine. Il fulcro dell'architettura è il meccanismo dell'attention, sfruttato da una struttura encoder-decoder costituita da uno stack di encoder e, in egual numero, uno stack di decoder. Ogni encoder è composto da un layer di *Multi-Head attention*, che permette al modello di stabilire le relazioni e i soggetti che ne prendono parte nella frase, seguito da una rete feed-forward che passa l'output all'elemento successivo. Per sviluppare il meccanismo di multi-head attention, vengono generate 3 matrici, *Query* (Q), *Key* (K) e *Value* (V), ottenute moltiplicando l'embedding in cui la parola in input viene tradotta per una matrice addestrata congiuntamente

al modello. Per ottenere lo score di ogni coppia di parole, viene effettuata la moltiplicazione tra Q (della prima parola) e K (della seconda), il risultato viene poi normalizzato e moltiplicato per V. Il primo strato dell'encoder codifica l'input nell'embedding, il primo strato del decoder codifica l'input ricevuto dall'encoder traslandolo di una posizione, questi aggiungono un vettore di *positional embedding* per incapsulare il concetto di posizionamento all'interno della sequenza.

L'output dell'ultimo encoder viene trasformato in un set di attention Key e Value, utilizzati poi nel layer di attention del decoder come descritto precedentemente, mentre l'output dell'ultimo decoder viene inviato al *Linear Layer* che proietta il vettore in uno ad alte dimensionalità, il *logits vector*, a cui viene applicato un calcolo di probabilità per determinare l'output secondo le possibilità offerte dal dizionario del modello.

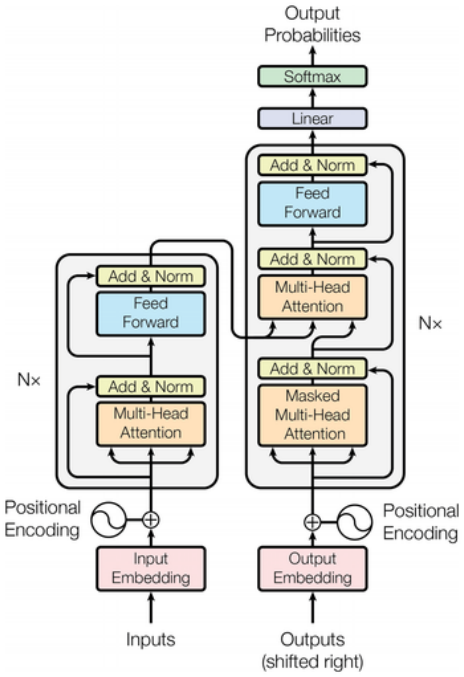

Figura 2.9: Architettura Transformer.
Fonte: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition

2.6 Metodo di Beam Search

Data la scarsa abilità dei modelli visti finora nel correggere le proprie previsioni, è stato disposto un metodo per tenere traccia delle $beam - width$ previsioni più promettenti. L'output prodotto dall'elaborazione compiuta dal modello ad un passo generico è un insieme di $beam - width$ token più promettenti, ottenuti mediante tecniche di calcolo di probabilità influenzate dalle precedenti previsioni migliori, iterando fino a che l'input non venga interamente processato, raggiungendo l'elaborazione ottimale.

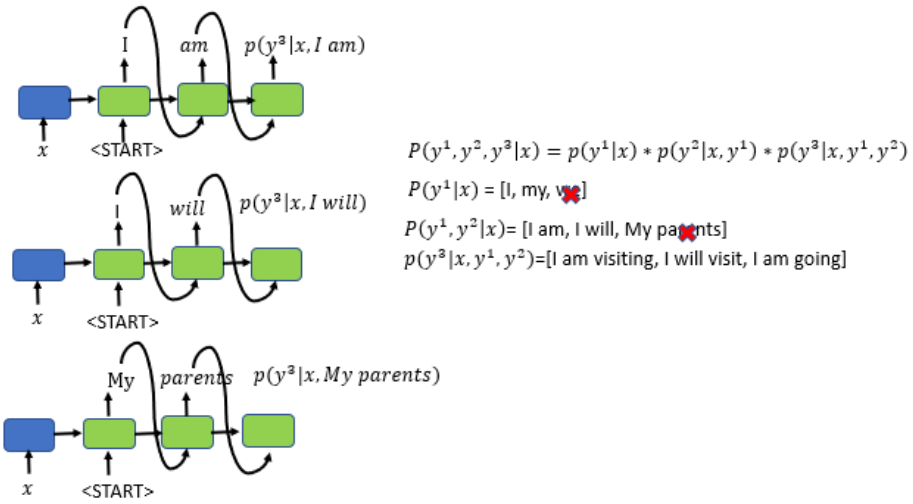

Figura 2.10: Modello che adotta Beam Search.
Fonte: https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f

Capitolo 3

Multi-Document Summarization

Il lavoro di questo elaborato è a supporto del problema di multi-document summarization, una specializzazione dell'automatic text summarization, in quanto applica il problema di ottenere una sintesi ad un numero di documenti maggiore di 1.

3.1 Automatic Text Summarization

L'automatic text summarization è un problema di NLP relativo all'estrazione, dato un testo, del contenuto informativo e della sua presentazione concisa.

Questa tematica risulta particolarmente attuale in molti ambiti sociali, durante la pandemia, ad esempio, le pubblicazioni in ambito medico hanno registrato un inaspettato aumento, arrivando ad una media di circa 200 al giorno.[1] Mantenendo regimi simili, il continuo aggiornamento del personale medico viene messo a dura prova e sistemi a supporto di schematizzazione e sintesi risultano nevralgici nell'alleggerire questo processo. Sono stati sviluppati principalmente due approcci in questo ambito:

- `Abstractive Summarization`, consiste nell'estrarre le informazioni importanti dal testo ed elaborare una nuova forma di presentazione, che risulti più concisa, generando un contesto;

- `Extractive Summarization`, consiste nel dividere il testo in parti, selezionare quelle di maggior interesse ed estrarle; il riassunto viene poi costituito dall'unione ordinata di queste.

[1] `https://www.recentiprogressi.it/archivio/3565/articoli/35454/`

Verranno successivamente presentati pregi e difetti di questi due approcci; sebbene l'abstractive summarization possa portare a risultati migliori, i limiti architetturali attuali han reso l'extractive la metodologia di riferimento, nonostante ultimamente si lavori per ottenere architetture ibride.

3.1.1 Extractive Summarization

Come descritto precedentemente, l'approccio di extractive summarization risulta più semplice e adattabile ad un modello meno complesso in quanto richiede meno elaborazione ed agisce come una sorta di classificatore, il cui compito è quello di etichettare le frasi in input come rilevanti o meno.

I passi generalmente adottati in questo approccio sono:

1. La costruzione di una `rappresentazione intermedia`, che permetta poi al modello di determinare la bontà informativa di ogni frase, ed estrarla di conseguenza;

2. Assegnare i punteggi di importanza alle frasi secondo una metrica stabilita;

3. Selezionare un numero di frasi sufficienti, scelto in ordine di importanza, ed eventualmente ri-costituire l'ordine originale.

La rappresentazione interna può essere generata seguendo due approcci distinti: la rappresentazione per `argomento`, che si basa appunto sull'individuazione degli argomenti principali, scelti secondo metriche di frequenza piuttosto che modelli Bayesiani e quella per `indicatori`, per cui le frasi vengono descritte dalle caratteristiche che incapsulano, ad esempio, la lunghezza o la centralità di posizione nel documento.

Rappresentazione per argomento

Per ottenere risultati soddisfacenti con questo tipo di rappresentazione, è fondamentale estrapolare gli argomenti trattati dal documento. L'estrazione dei suddetti può avvenire ad esempio fissando una soglia al di sopra della quale, contando il numero di occorrenze di una parola in un testo, questa viene considerata un argomento, mentre altri modi coinvolgono la stima di metriche di similarità e il conteggio delle occorrenze [6] che vanno a determinare le *topic signature* [7].

Dopo aver stabilito gli argomenti, le frasi vengono sottoposte o ad un conteggio del numero di argomenti nelle frasi (rischia di sbilanciarsi verso le frasi più lunghe), oppure ad una stima di correlazione semantica con questi.

Determinare il punteggio da assegnare ad una frase dipende dalla rappresentazione intermedia scelta: nel caso di quella centrata sugli argomenti, è

necessario stimare quanto arricchisce la conoscenza su questi, mentre nel caso della centralità degli indicatori, vengono assegnati dei pesi alle caratteristiche prese in esame, moltiplicati per le loro misurazioni frase per frase. Al fine di stimare i pesi degli indicatori è buona norma integrare dei metodi di machine learning a supporto.

I parametri dei pesi possono essere valori continui su un intervallo, oppure binari sui valori 0 e 1 e sono determinati secondo diverse tecniche:

- *word probability*, dove l'importanza di un termine viene calcolata come

$$P(w) = \frac{f(w)}{N}$$

  dove $P(w)$ rappresenta l'importanza di una parola w, $f(w)$ il numero di occorrenze in una frase e N la lunghezza della frase;

- `TF-IDF`, come descritto precedentemente, il peso di una parola viene determinato in base al numero di occorrenze in un documento e, agendo in maniera inversa, il numero di documenti in cui la parola appare:

$$q(w) = f_d(w) * log\frac{|D|}{f_D(w)}$$

  dove $f_d(w)$ è la frequenza della parola w nel documento d, $f_D(w)$ è il numero di documenti in cui w appare e $|D|$ è il numero di documenti presi in considerazione. L'applicazione della funzione *log* fa si che qualora una parola appaia in tutti i documenti, l'argomento diventi 1 e di conseguenza il peso della parola sia $log(1) = 0$. I risultati ottenuti possono essere presi direttamente come pesi, vi sono però esempi di ulteriori elaborazioni [8];

- `Latent Semantic Analysis` [9] è un metodo non supervisionato per rappresentare la semantica di un testo. Prevede la costruzione di una matrice costituita dalle parole e dalle frasi, le celle contengono il punteggio ottenuto come TF-IDF, a cui viene poi applicato il principio di *Single Value Decomposition*, secondo cui una matrice è scomponibile nel prodotto di 3 matrici: $A = U\Sigma V^T$, dove U esprime i pesi delle parole rispetto agli argomenti, Σ è la matrice dei pesi di ogni argomento e V^T contiene i pesi argomenti - frasi.

Rappresentazione per Indicatori

Mirano a rappresentare il testo come insiemi di caratteristiche, utilizzate per classificare le frasi direttamente, vengono individuati due approcci:

- Metodi basati su `grafi`, rappresentano il documento come struttura a grafo, dove le frasi rappresentano i nodi, mentre gli archi una metrica di similarità. I sotto-grafi che si vanno a generare costituiscono gli argomenti trattati, e i nodi più connessi all'interno di questa sotto-struttura le frasi più importanti;

- Metodi basati su machine learning, modellano il problema approcciandolo come una classificazione, etichettando le frasi come appartenenti al riassunto o meno, basandosi sui punteggi degli indicatori.

Selezione delle frasi

La successiva selezione delle k frasi più importanti avviene per mezzo di due tipologie algoritmi:

- Algoritmi `greedy`, che selezionano le frasi ordinandole per importanza secondo la metrica utilizzata;

- Approccio di ottimizzazione, alla base della scelta delle frasi si pone il vincolo di ottenere il miglior contenuto informativo spaziato su tutti gli argomenti, andando a minimizzare eventuali ripetizioni.

Altri fattori influenzano l'ottimalità delle scelte, tra cui ambito del documento e il tipo.

3.1.2 Abstractive Summarization

L'abstractive summarization è l'approccio che prevede l'estrazione delle informazioni dal documento e l'elaborazione di una nuova rappresentazione in cui riportare il contenuto informativo estratto, per cui viene introdotto un modello generatore. Sebbene questo sia più simile all'approccio umano nella risoluzione del problema di sintesi, la difficoltà nel generare un contesto che renda la nuova rappresentazione presentabile risulta frenare molto questo metodo; la ricerca ha però condotto passi avanti in questa direzione, portando diversi modelli.

Structure-based

Questa famiglia di metodologie prevede l'estrazione dell'informazione e una successiva ricostruzione di contesto mediante strutture opportune. Prodotto quindi il testo relativo all'informazione da tradurre, viene costituito un parser per la conversione in una delle strutture dati proposte:

- `Tree-based`, per cui il testo che contiene le informazioni sul documento viene convertito in una struttura ad albero mantenendo l'aspetto di similarità originale;
- `Template-based`, prevede l'utilizzo di template pre-definiti per generare il riassunto finale, utilizzando frammenti di testo originario estratto secondo parole chiave;
- `Ontology-based`, si basa sul concetto di ontologia, che incapsula gli argomenti chiave, la loro definizione e le loro relazioni. Un esempio di ontologia nota è WordNet [10]. Questa divisione risulta essere molto utile in caso di modifiche dei documenti, per la disambiguazione di termini e per l'eliminazione della ridondanza;
- `Graph-based`, opta per la rappresentazione a grafi. Le frasi simili vengono fuse insieme per formare strutture che incapsulano il concetto di similarità ed eliminano la ridondanza;
- `Rule-based`, vengono fornite regole e categorie al modello che estrapola di conseguenza le frasi importanti. Dopo una fase di categorizzazione del testo viene ricostruito un contesto andando a rispondere a domande pre-definite passate al sistema.

Semantic-based

Alla base di questo approccio c'è l'estrazione di strutture predicato-argomento o, in alternativa, grafi semantici prendendo in esame elementi ritenuti informativi. La rappresentazione risultante viene poi fornita ad un `Natural Language Generation system`, che genera il riassunto finale.

- `Information item based`, estrae dal documento unità informative, ovvero i più piccoli concetti informativi, composti da entità, caratteristiche e relazioni con altre entità; le entità emergono elaborando le frasi più importanti;
- `Predicate-argument based`, estrae i predicati intesi come strutture soggetto, verbo e complementi. Successivamente, i predicati considerati

simili vengono fusi, da questa unione vengono estratte le caratteristiche di spicco, a cui viene assegnata un'importanza. I predicati più importanti vengono poi utilizzati dal sistema di generazione per ottenere il riassunto finale;

- `Semantic graph based`, propone la rappresentazione di relazioni semantiche, basate sull'ontologia del testo, mediante struttura a grafi. Le strutture a grafi sono di diverso tipo:
  - Rich Semantic Graphs, nomi e verbi, ottenuti per mezzo di parser e ontologie, rappresentano i nodi, mentre gli archi sono le relazioni semantiche e topologiche tra questi;
  - `Abstract Meaning Representation`, sono grafi etichettati, orientati e aciclici, che offrono la rappresentazione semantica di una frase. I nodi sono costituiti da concetti e gli archi sono le relazioni tra questi. Questa metodologia elabora molta dell'informazione contenuta in una frase, tra cui le named entities, ed elimina la ridondanza fondendo i concetti simili. Verrà approfondita la tecnica successivamente (4.3.1);
  - Basic Semantic Unit Based, per ovviare alla scarsa compatibilità della rappresentazione AMR con i sistemi di generazione, questo approccio consiste nel generare reti semantiche centrate sul concetto di attore e ricettore, per generare in maniera efficiente il riassunto finale.
- `Multimodal semantic method`, permette l'estrazione di informazioni e conoscenza da fonti non prettamente testuali, tra cui immagini e video. Non sono però forniti strumenti a supporto della valutazione della bontà di questo metodo.

Approcci basati su deep learning e reti neurali

Dato l'importante impatto dei sistemi basati su deep learning nel mondo dei problemi NLP e l'abilità di estrarre le features in maniera autonoma, vengono sfruttate diverse architetture per lavorare anche in ambito sintesi.

- Modelli Encoder-Decoder, descritti nella sezione 2.4.5
- Celle LSTM e GRU, descritte rispettivamente nelle sezioni 2.4.2 e 2.4.3
- Transformers, descritti nella sezione 2.5

Sfruttando queste architetture sono stati raggiunti risultati che hanno superato il precedente stato dell'arte. I modelli più gettonati in quest'ambito sono del tipo *seq2seq*, ovvero, processano una sequenza di input ed ottengono una sequenza di output.

BERT

Acronimo di `Bidirectional Encoder Representation from Transformers` [11], è un modello basato su transformers utilizzato per l'elaborazione del linguaggio naturale. Rilasciato dal team Google, è stato sviluppato in 2 varianti, base e large, che differiscono nel numero di encoder e sistemi di self-attention che l'architettura adotta. Per essere utilizzato in task di generazione, necessita di integrazione di decoder, uno dei più utilizzati è GPT [12].

Ciò che rende questi modelli differenti è l'adozione di tecniche di pre-training differenti, nel caso di BERT ad esempio, vengono predisposte 3 fasi:

- Il primo obiettivo è quello di comprendere il linguaggio e l'utilità del context, per cui vengono impiegate attività di `Masked Language Model`, in cui il modello è chiamato a predire parole rimosse da frasi comuni, e di `Next Sentence Prediction`, per cui il sistema deve riconoscere relazioni tra frasi, ad esempio se una frase è diretta conseguenza di un'altra;

- L'elaborazione simultanea di più frasi, per cui vengono predisposte task multiple su più frasi;

- Sistema di embedding, che permette di elaborare più frasi insieme. Si compone di token, che mappa una parola in un vettore, a cui viene aggiunto l'embedding di segment e position, che incapsulano il concetto di ordine e posizione nel vettore.

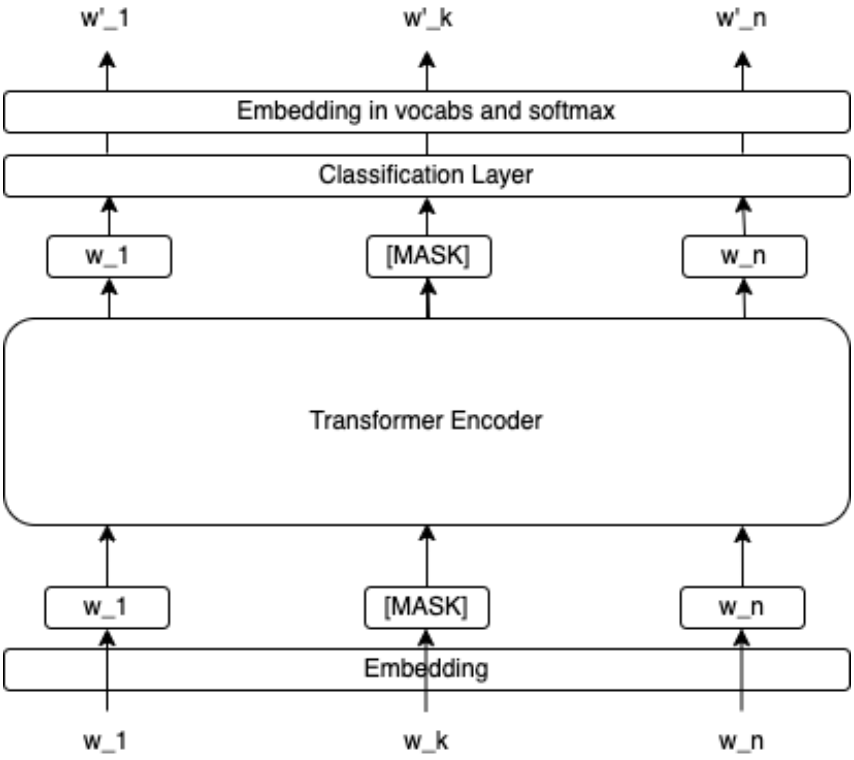

Figura 3.1: Architettura di BERT.

BART

È un modello [13] di *denoising autoencoder*, costituito da un `Bidirectional encoder` e un *left-to-right* autoregressive decoder, che registra prestazioni allo stato dell'arte per molti problemi di NLP, tra cui machine traslation e sequence generation. Addestrato come tale è particolarmente utile nella ricostruzione di testo corrotto, ciò lo rende affine a task di generazione di testo. Seguendo la grafica di funzionamento (3.2), l'encoder bidirezionale agisce nel costruire l'embedding della frase in input in cui 2 token vengono mascherati, una volta inviato al decoder, questo elabora e produce una frase simile a quella fornita in input.

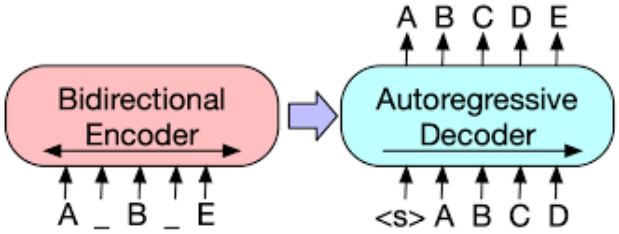

Figura 3.2: Funzionamento di BART.
Fonte: https://paperswithcode.com/method/bart

Utilizzo dei modelli

Le architetture vengono fornite in stato di pre-training, al fine di utilizzarle in un ambito specifico è buona norma attuare una fase di fine-tuning, specializzando il modello su un task particolare e una lingua specifica, al fine di ottenere risultati più accurati possibile. Per questo compito è necessario raccogliere dataset specializzati, spesso risultati di onerosi lavori preliminari.

3.1.3 Metriche di valutazione

La valutazione della bontà di una sintesi è un compito difficile, anche gli esperti del settore non trovano un punto di accordo generale. Una possibile valutazione potrebbe essere ottenuta con l'impiego di persone fisiche che valutano aspetti di copertura degli argomenti chiave e di domande pre-impostate.

Per ottenere invece una valutazione automatizzata, sono state introdotte metriche di riferimento.

ROUGE

Acronimo di *Recall Oriented Understudy for Gisting Evaluation*, è un metodo di valutazione sintattico e di fatto la metrica più diffusa; confronta un riassunto fornito, in questo caso prodotto da un modello, con uno *target*, che agisce come sintesi ideale, generalmente prodotto preliminarmente per mezzo di elaborazione umana. Si divide in diverse sotto-metriche, per cui si ottengono valori di:

- *Precision*, definita come il rapporto tra le istanze rilevanti e il numero di istanze ottenute;
- *Recall*, definita come la frazione di istanze rilevanti ottenute;
- *F-measure*, è una combinazione di precision e recall, ottenuta come

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

Le metriche più gettonate sono quelle di ROUGE-N e ROUGE-L, ma ne esistono altre:

- `ROUGE-N`, calcola la percentuale di n-gram appartenenti al riassunto target che appartengono anche al riassunto da analizzare;

- `ROUGE-L`, si basa sul concetto di *Longest Common Sequence*, secondo cui tanto è più lungo l'n-gram appartenente ad entrambe i testi, tanto più simili questi saranno;

- `ROUGE-SU`, chiamata *skip bi-gram and uni-gram ROUGE*, considera sia gli uni-gram sia i bi-gram ammettendo parole divise da un numero arbitrario di parole;

- `ROUGE-WE`, aggiunge alla metrica il concetto di embedding. Viene stabilita una metrica basata sulla semantica in quanto misura la similarità tra le frasi proiettate in embedding.

BLEU

Acronimo di *Bilingual Evaluation Understudy Score*, è un metodo di valutazione sintattico simile alla metrica ROUGE precedentemente descritta, prevede infatti il conteggio degli n-gram presenti nel riassunto candidato che sono presenti anche nel riassunto target, come ROUGE non tiene conto dell'ordine delle parole. Generalmente il punteggio ottenuto per un n-gram viene normalizzato in base al numero di occorrenze registrate.

3.2 Multi-Document Summarization

Come precedentemente descritto, il problema di multi-document summarization è un caso particolare di automatic text summarization, in cui il testo da riassumere non è più unico, ma si compone di più documenti.

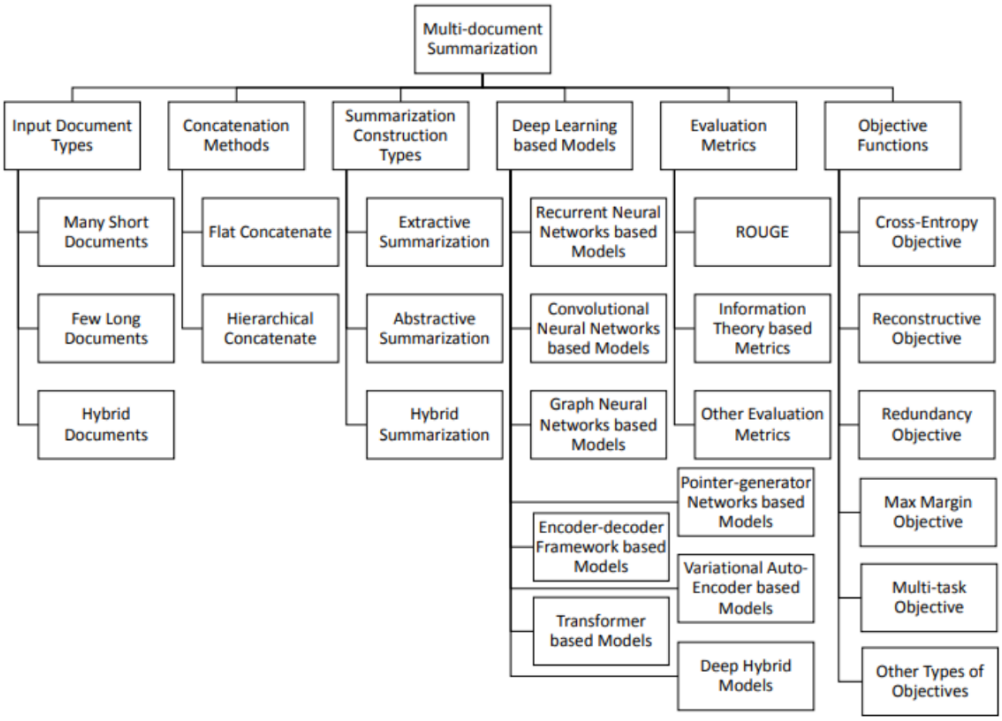

Figura 3.3: La struttura gerarchica di MDS.
Fonte: [14]

Le tipologie di documenti in input differiscono per struttura e tipologia di concatenazione, come riportato in figura 3.3, le modalità di concatenazione sono:

- `Concatenazione piatta`, dove i documenti vengono separati da token; questa metodologia permette un'elaborazione piatta dei documenti, convertendo il problema di MDS ad uno di SDS di grosse dimensioni, è necessario adottare un modello che possa elaborare grosse dimensioni di documenti;

- `Concatenazione gerarchica`, deve mantenere le relazioni tra documenti. Solitamente viene adottata una rappresentazione mediante strutture a grafo, che preservano le relazioni.

Gli approcci di sintetizzazione sono gli stessi descritti precedentemente, viene però privilegiata una scelta ibrida tra abstractive ed extractive summarization:

- Approccio `Extractive - Abstractive`, per cui vengono estratte le frasi ritenute più importanti, utilizzate poi per generare il riassunto finale;

- Approccio `Abstractive - Abstractive`, per cui vengono combinati più riassunti di tipo astrattivo, il risultato viene ulteriormente riassunto ed utilizzato per generare il riassunto finale.

3.2.1 Modelli di MDS

Ci sono diverse architetture in ambito MDS che permettono di ottenere risultati migliori in condizioni differenti.

- `Recurrent Neural Network based`, si prestano a lavorare meglio con dati di tipo sequenziale, in quanto estraggono relazioni sequenziali. Non si prestano al parallelismo e a processare lunghi documenti;

- `Convolutional Neural Network based`, sfruttando i convolutional kernel, possono processare il testo dopo averlo trasformato in vettore. Applicate al MDS, offrono la possibilità di rappresentazione semantica;

- `Graph Neural Network based`, possono modellare in maniera molto efficace relazioni di tipo semantico e sintattico. Vengono integrate a modelli DNN per generare gli embedding dei documenti;

- `Pointer-generator Networks Based`, sono utilizzati per far fronte a problemi di errori e ridondanza. Introducono il metodo di *Maximal Marginal Relevance* che seleziona un insieme di frasi dai documenti sulla base di massimizzazione dell'informazione e al contempo minimizzazione della ridondanza; queste frasi vengono aggiunte al riassunto in ordine di importanza fino al raggiungimento di una soglia;

- `Transformer Based`, vantaggioso in termini di parallelismo, riesce inoltre a gestire lunghe sequenze di testo;

- `Modelli ibridi`, combinano più tecnologie per sfruttare i punti di forza di ognuna di queste.

3.2.2 Funzioni obiettivo

Per poter adattare e addestrare i modelli su una task precisa è cruciale individuare un modo per definire una metrica di errore commesso, al fine di minimizzarlo ottenendo la miglior combinazione di parametri che costituiscono il modello. Con questo fine, sono state studiate ed adattate diverse funzioni dette obiettivo che definiscono l'errore commesso in particolari tipologie di applicazioni.

- `Cross-Entropy`, misura la distanza tra due distribuzioni, in MDS misura la distanza di distribuzione tra il riassunto generato e il riassunto target.

$$L = -\sum_{i=1} y_i * \log(z_i)$$

  Dove y_i rappresenta il vettore ottenuto dalla frase del riassunto target, mentre z_i quello della frase del riassunto generato;

- `Redundancy objective`, utilizzata per minimizzare la sovrapposizione di unità semantiche nel riassunto generato, e massimizzare la copertura informativa.

$$L = Sim(x_i, x_j)$$

  Dove la funzione Sim misura la sovrapposizione di due elementi, ovvero frasi, argomenti o documenti;

- `Multi-Task objective`, combina funzioni obiettivo differenti per ottenere risultati in più ambiti.

$$L = L_{summ} + L_{other}$$

Capitolo 4

Progetto

4.1 Introduzione

Il progetto verte sull'analisi di metodologie di soft labeling del testo elaborato nel contesto della multi-document summarization. Il modello che elabora il riassunto impone un limite massimo in termini di lunghezza del testo passatogli; avendo adottato come dataset di riferimento Multi-LexSum [15], la mole di testo da processare risulta molto elevata, per cui viene utilizzato il soft labeling, il cui obiettivo è quello di estrarre le frasi più informative dai documenti in modo da rimuovere eventuale rumore, e passarle successivamente al modello che si occuperà di generare un riassunto da confrontare poi con il target. Come suggerisce il nome, le etichette di importanza derivate dal processo di soft labeling vengono generate tramite euristiche e non in maniera supervisionata da esperti del settore, pertanto non rappresentano il cosiddetto *ground truth*. Al fine di simulare un caso di studio reale, è stata scelta la dinamica *low-resource*, in cui si ha a disposizione un quantitativo di input limitato ma sufficiente per ottenere risultati qualitativamente affidabili.

4.2 Dataset

Il dataset preso in esame è Multi-LexSum [15], una raccolta di oltre 40000 documenti, di lunghezza media 75000 parole, riguardanti casi reali in ambito legale tratti da *Civil Rights Litigation Clearinghouse*, con allegati circa 9000 riassunti astrattivi elaborati da esperti del settore. L'obiettivo della summarization in ambito legale è di produrre articoli che catturino i principali dettagli e descrivano lo storico del contenzioso del caso adottando un linguaggio comprensibile, il tutto presentato in maniera concisa. L'ambito CRLC richiede la produzione di riassunti a diversi livelli di `granularità`:

- *Tiny*, per cui si producono riassunti di circa 25 parole, con l'obiettivo di descrivere il caso in uno specifico punto;

- *Short*, per cui si producono riassunti di circa 130 parole, con cui si fornisce una descrizione breve del contesto, delle parti in causa e il risultato del caso;

- *Long*, per cui si producono riassunti di circa 600 parole, si articola in più paragrafi, in cui, oltre alla descrizione del contesto e delle parti in causa, compaiono procedure, eventi e risultati.

Questa granularità fornisce supporto ad analisi più o meno profonde.

Tutto il materiale utilizzato nel dataset è stato curato da esperti del settore. L'iter seguito dagli esperti per la definizione di ogni documento è il seguente:

1. Dato un assegnamento specifico, l'esperto legge i documenti prodotti del caso, ed estrapola quelli di maggior interesse;

2. Per assicurarsi di fornire tutte le importanti informazioni del caso, l'esperto può ricorrere ad una lista di fatti da includere nel riassunto. Generalmente viene scritto prima il riassunto long, da cui poi vengono estratti anche lo short e il tiny. Data la dilatazione dei tempi per la risoluzione dei casi legali, i documenti e di conseguenza i riassunti, potrebbero essere soggetti ad aggiornamenti;

3. Una volta prodotto l'insieme dei riassunti viene passato, per un controllo ulteriore, ad un altro esperto, che può editarlo all'occorrenza. Successivamente, il riassunto revisionato viene aggiunto al dataset generale.

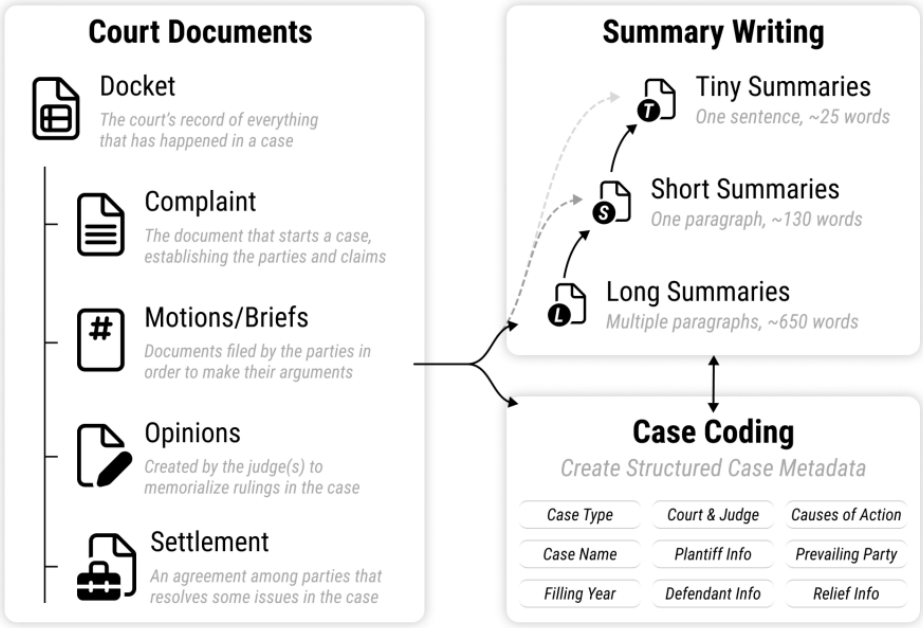

Figura 4.1: Piano di lavoro per ottenere la sintesi.
Fonte: [15]

4.3 Metriche

Per ottenere una panoramica più ampia possibile, sono state adottate diverse metriche per il soft labeling, di natura sintattica, semantica e applicate a strutture di rappresentazione differenti:

- `ROUGE`, descritta nella sezione 3.1.3
- `BLEU`, descritta nella sezione 3.1.3
- `BERTScore`, descritta nella sezione 4.3.2
- `Smatch`, descritta nella sezione 4.3.3
- `WWLK`, descritta nella sezione 4.3.4

4.3.1 AMR

Acronimo di `Abstract Meaning Representation` [16], è un linguaggio di rappresentazione semantica che astrae dalla rappresentazione sintattica. La struttura a grafi utilizzata per la rappresentazione è orientata, etichettata e aciclica, dove i nodi caratterizzano i concetti e gli archi le relazioni tra

questi. Data la sua flessibilità, è stata impiegata in diversi ambiti NLP, tra cui la *machine translation*. Per la traduzione dei documenti, è stata utilizzata l'architettura proposta da `SPRING` [17], che impiega BART (3.1.2) con un vocabolario modificato per poter lavorare con i simboli AMR.

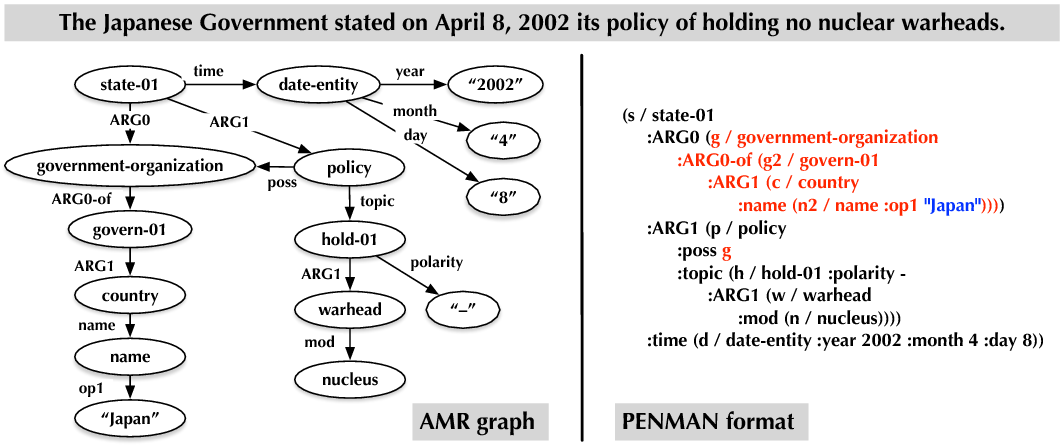

Figura 4.2: Esempio di rappresentazione AMR e relativo encoding.
Fonte: https://www.semanticscholar.org

4.3.2 BERTScore

È un metodo di valutazione semantico per stabilire la similarità tra testi. Entrambi i riassunti in questo caso, target e *source*, vengono passati ad un modello BERT (3.1.2) per ottenere gli embedding su cui verrà calcolata la similarità parola per parola, per ognuna di queste viene preso il punteggio migliore, su cui viene successivamente calcolata precision, recall e f-score.

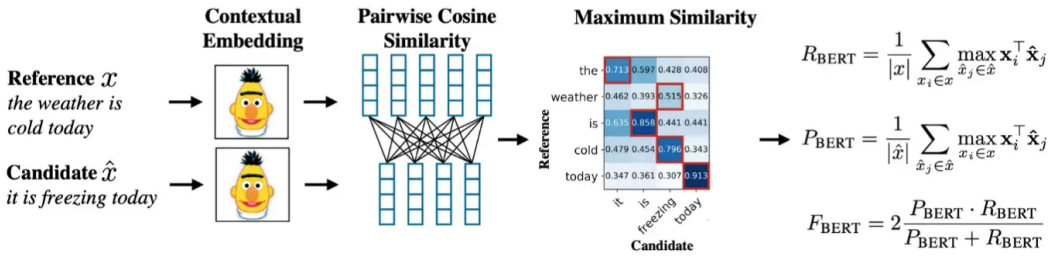

Figura 4.3: Calcolo di BERTScore.
Fonte: https://towardsdatascience.com/bertscore-evaluating-text-generation-with-bert-beb7b3431300

4.3.3 Smatch

È un metodo di valutazione [18] applicato alle *semantic feature structure representation*, ad esempio la rappresentazione *AMR* (4.3.1). Dati due grafi AMR in input, tutti i nodi del primo vengono corrisposti ai nodi del secondo sulla base della condivisione di concetti (qualora non ce ne siano vengono abbinati casualmente) e, mediante programmazione lineare intera, viene massimizzato il numero di triple che si combinano perfettamente, dove per tripla si intende un insieme di 3 dati costituiti da *relazione(variabile*$_1$*, variabile*$_2$*)* oppure *relazione(variabile, concetto)*. Il numero massimo di triple perfettamente abbinate viene poi utilizzato per ottenere il punteggio smatch secondo le metriche di precision, recall e f-measure.

4.3.4 Weisfeiler Leman metric

È un metodo di valutazione [19] applicato alle *semantic feature structure representation*, ad esempio la rappresentazione *AMR* (4.3.1). Per calcolare la similarità tra strutture a grafi, questo metodo parte con l'elaborazione degli embedding per ogni grafo, proiettandolo in uno spazio che permetta di descrivere vari livelli di contestualizzazione dei nodi. Questi embedding vengono poi confrontati facendo emergere un costo minimo per trasformare un grafo nell'altro, che rappresenta la distanza di similarità tra i grafi.

4.4 Modello PRIMERA

Dopo la fase di soft labeling, il testo viene estratto e passato al modello per la generazione del riassunto da confrontare con quello target. Il modello utilizzato in questa fase è PRIMERA [20], che ha registrato i risultati stato dell'arte per la multi-document summarization sul dataset multi-lexsum, utilizzando un algoritmo a struttura piramidale; il primo passo dell'esecuzione prevede l'estrazione delle *Summary Content Unit*, ovvero le informazioni di rilievo presenti nei documenti, di queste viene effettuato il conteggio dei documenti in cui l'argomento viene trattato, costituendo una classifica ordinata in maniera decrescente rispetto a questa occorrenza; il riassunto finale viene poi formato andando a ricostruire le frasi che riguardano gli argomenti seguendo l'ordine di importanza stabilito precedentemente.

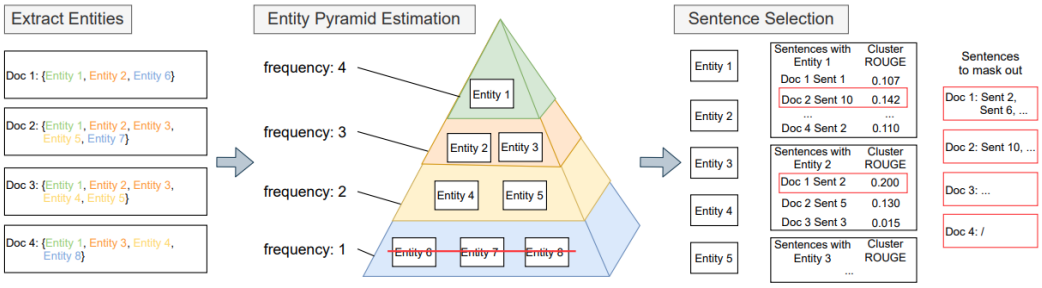

Figura 4.4: Algoritmo piramidale.
Fonte: [20]

Il fulcro della struttura è il modello LED [21], che elabora l'input e mantiene la separazione dei documenti aggiungendo un simbolo speciale.

4.5 Tecnologie utilizzate

4.5.1 Python

Python è un linguaggio di programmazione *multi-paradigma* ad alto livello rilasciato all'inizio degli anni '90. È *object-oriented* e supporta la programmazione strutturale e funzionale, ha inoltre la peculiarità di essere un linguaggio interpretato che conferisce la portabilità su più piattaforme, le quali necessitano di un interprete. Un'ampia *standard library* e la grande disponibilità di librerie ha fatto sì che Python sia diffuso in vari ambiti, tra cui applicazioni web, *computer vision*, *data science* e intelligenza artificiale.

4.5.2 Jupyter Notebook

Jupyter Notebook è un'applicazione web open-source che permette di creare e condividere documenti testuali interattivi costituiti da celle contenenti codice sorgente eseguibile, celle di testo, immagini o altri elementi ancora. Supporta l'esecuzione di applicazioni Scala e Python su piattaforme big data, integrando Apache Spark. Presenta un'interfaccia semplice ed intuitiva:

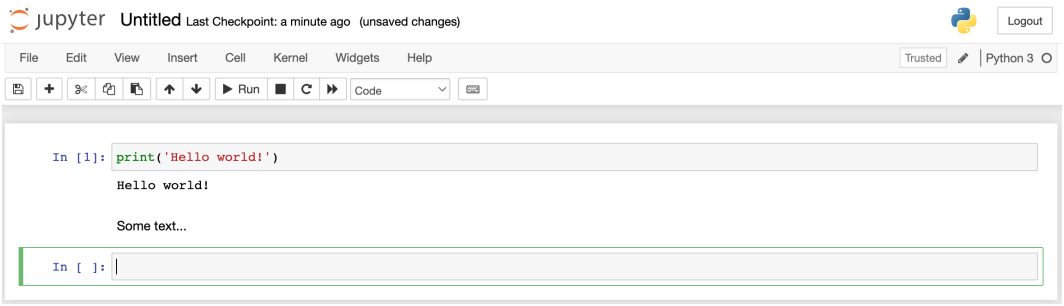

Figura 4.5: Interfaccia Jupyter Notebook.

4.5.3 Anaconda

Anaconda è una distribuzione di Python con l'obiettivo di semplificare le gestione dei *package* e la loro installazione. Al suo interno si trova `conda`, un *package manager* open-source cross-platform con integrato un sistema di gestione dell'ambiente, che in fase di installazione di un pacchetto analizza tutto ciò che è già installato, si occupa di recuperare la versione richiesta e all'occorrenza solleva errori in caso di incompatibilità. È possibile creare degli ambienti che contengano configurazioni di pacchetti differenti e navigare rapidamente tra questi.

4.5.4 Hugging Face

Hugging Face[1] è un provider open-source che fornisce librerie e dataset multi-purpouse, particolarmente gettonato in ambito NLP. Con un'interfaccia molto semplice è possibile, previa installazione, utilizzare alcune metriche di valutazione, tra cui BLEU (3.1.3) e Bertscore (4.3.2), oltre che dataset predisposti per addestramenti e valutazioni e modelli pre-addestrati.

4.5.5 GitHub

GitHub[2] è un servizio di hosting internet per sviluppo di software open-source e *version control*. Permette uno sviluppo di software fluente e ben strutturato, con la possibilità di tracciare e gestire le modifiche al progetto e di condividerle con altri collaboratori.

[1] `https://huggingface.co/`
[2] `https://github.com/`

4.5.6 Slurm

Acronimo di *Simple Linux Utility for Resource Management*, è un *job scheduler* per kernel Linux e Unix-like, utilizzato in ambito *hyper-computing*. Permette di:

- Allocare l'accesso, esclusivo o non, alle risorse di calcolo per una durata di tempo variabile;
- Fruire di un framework per eseguire e monitorare i progressi di un programma su un sistema di nodi;
- Gestire eventuali contese di risorse con code di attesa dei processi da eseguire.

4.5.7 Docker

Docker[3] è una piattaforma software che permette di creare, testare e distribuire applicazioni con rapidità. Si basa sul concetto di *container*, un ambiente isolato standard che virtualizza il sistema operativo rendendosi portatile ed eseguibile su più piattaforme; offre librerie, strumenti di sistema, codice e runtime per eseguire il software.

4.6 Codice prodotto

Il programma si divide in due parti:

1. Vengono scanditi tutti i documenti forniti in input e viene assegnato un punteggio ad ogni frase, ottenuto come la massima similarità tra quelle calcolate con ciascuna frase del riassunto target;
2. Viene ricostruito il testo da passare al modello per l'elaborazione del riassunto, con l'aggiunta dei simboli che rappresentano la fine di un documento.

Per quanto riguarda le metriche di Smatch (4.3.3) e wwlk (4.3.4), è necessario predisporre la rappresentazione AMR (4.3.1) per cui viene utilizzato un modello già addestrato, che presenta problemi di incompatibilità con il modello PRIMERA. Il modello che si occupa della rappresentazione AMR necessita di installazione di una versione della libreria transformers = 2.11, per cui va eseguito un downgrade rispetto alla versione 4.24 aggiornata, che richiede PRIMERA.

[3] `https://www.docker.com/`

L'esecuzione delle due metriche basate su rappresentazione AMR è risultata quindi problematica, ed è stato necessario dividere in 2 la computazione, la prima computazione calcola i risultati di similarità delle frasi convertite in rappresentazione AMR e le trascrive su file, la seconda computazione, con la versione della libreria transformers aggiornata, legge i risultati di similarità da file e costruisce il nuovo testo da passare a PRIMERA.

Preliminarmente al calcolo dei punteggi di similarità, vengono scanditi tutti i dataset in input e viene assegnata ad ogni testo la sua lunghezza calcolata per mezzo del tokenizer del modello; questo permette di saltare il soft labeling nel caso in cui la lunghezza del testo convertito in token non superi la lunghezza massima che il modello può processare.

In questa fase, viene costruito anche il percorso in cui salvare eventuale pre-processing del testo e i risultati.

4.6.1 Estrazione dei punteggi di similarità

I punteggi di similarità vengono calcolati dalla funzione organizzata come segue:

```
def soft_labeling(args, dataset, text_column, summary_column,\
    len_column, dataset_type):
    # Allocazione delle strutture utilizzate per calcolare la\
    # metrica di similarità richiesta
    # ...
    # Iterazione su ogni testo dei documenti in input
    # ...
    # Eventuale pre-processing del testo (rappresentazione AMR)
    # ...
    # Calcolo della similarità
    # ...
    # Restituzione dei risultati
```

Questa, viene richiamata per ogni parte del dataset, training, evaluation e test; i risultati vengono salvati in strutture dati differenti. Per fattori di ottimizzazione, le strutture utilizzate per calcolare la metrica di similarità richiesta vengono allocate prima di iterare sui testi dei documenti da processare, mentre vengono poi utilizzate concretamente nella fase di calcolo della similarità.

Un esempio di funzione per il calcolo della similarità con metrica smatch:

```
# Eventuale pre-processing del testo

# Costruzione delle rappresentazioni AMR
```

```
def prepare_amr(dataset, summary, paths, hyperparams, model,\
    tokenizer, device, num):
    import torch
    from spring_amr.penman import encode
    def read_file_in_batches(path, batch_size, max_length=100):
        data = []
        idx = 0
        for line in path:
            if char_DS not in line:
                line = line.strip()
                if not line:
                    continue
                n = len(line.split())
                if n > max_length:
                    continue
                data.append((idx, line, n))
                idx += 1

        def _iterator(data):
            data = sorted(data, key=lambda x: x[2], reverse=True)
            maxn = 0
            batch = []
            for sample in data:
                idx, line, n = sample
                if n > batch_size:
                    if batch:
                        yield batch
                        maxn = 0
                        batch = []
                    yield [sample]
                else:
                    curr_batch_size = maxn * len(batch)
                    cand_batch_size = max(maxn, n) * (len(batch)+1)

                    if 0 < curr_batch_size <= batch_size and\
                        cand_batch_size > batch_size:
                        yield batch
                        maxn = 0
                        batch = []
                    maxn = max(maxn, n)
                    batch.append(sample)
```

```
45              if batch:
46                  yield batch
47
48          return _iterator(data), len(data)
49
50      def get_amr_from_file(sentences, model, hyperparams,\
51          summary=False):
52          with tqdm(desc=f"preparing amr {num}", total=len(sentences))\
53              as bar:
54              for idx, sent in enumerate(sentences):
55                  if char_DS in sent:
56                      sent = sent.replace(char_DS, '')
57                  if sent != '':
58                      x = tokenizer.batch_encode_plus([sent\
59                          .replace('\n', '')], return_tensors='pt',\
60                              max_length=128)
61                      x = {k: v.to(device) for k, v in x.items()}
62                      with torch.no_grad():
63                          model.amr_mode = True
64                          out = model.generate(**x, max_length=256,\
65                              decoder_start_token_id=0,\
66                               num_beams=hyperparams['beam_size'])
67                      bgraphs = []
68                      graph, status, _ = tokenizer.decode_amr(
69                          out.tolist()[0], restore_name_ops=\
70                              hyperparams['restore_name_ops'])
71                      if 'OK' not in str(status):
72                          continue
73                      graph.metadata['status'] = str(status)
74                      graph.metadata['nsent'] = str(idx)
75                      graph.metadata['snt'] = sent.replace('\n', '')
76                      bgraphs.append((idx, graph))
77                      bgraphs.sort(key=lambda x: x[0])
78                      f = open(f'{paths["summary_path"]}/amr_summary_\
79                          {num}.txt', 'a') if summary else\
80                              open(f'{paths["source_path"]}/amr_source_\
81                                  {num}.txt', 'a')
82                      for i, g in bgraphs:
83                          f.write(encode(g) + '\n\n')
84                  bar.update(1)
85
```

```
86      get_amr_from_file(dataset, model, hyperparams)
87      get_amr_from_file(summary, model, hyperparams, summary=True)
88
89
90  if args.amr_smatch:
91  # Allocazione delle strutture utilizzate
92
93      # Aggiungo i moduli utilizzati per la rappresentazione AMR
94      if 'spring' not in sys.path:
95          sys.path.append(module_path + '/spring')
96      if 'amr-utils' not in sys.path:
97          sys.path.append(module_path + '/amr-utils')
98      if 'weisfeiler-leman-amr-metrics/src' not in sys.path:
99          sys.path.append(module_path + '/weisfeiler-leman-amr-\
100             metrics/src')
101
102     # Vengono utilizzati import inline per sopperire all'\
103     # incompatibilità del modello di rappresentazione AMR per la\
104     # libreria transformers
105     import torch
106     from spring_amr.penman import encode
107     from spring_amr.utils import instantiate_model_and_tokenizer
108     Path(f'{module_path}/datasets/amrs/{args.dataset}/{dataset_type}\
109         /amrs_summary').mkdir(parents=True, exist_ok=True)
110     Path(f'{module_path}/datasets/amrs/{args.dataset}/{dataset_type}\
111         /amrs_source').mkdir(parents=True, exist_ok=True)
112     # Raccolti i percorsi in cui salvare le rappresentazioni AMR\
113     # e i parametri passati al modello
114     paths = dict([
115         ('summary_path', f'{module_path}/datasets/amrs\
116             /{args.dataset}/{dataset_type}/amrs_summary'),
117         ('source_path', f'{module_path}/datasets/amrs/\
118             {args.dataset}/{dataset_type}/amrs_source'),
119         ('metrics', f'{module_path}/datasets/amrs/metrics.txt')
120     ])
121     # Vengono raccolti tutti i parametri utilizzati dal modello
122     hyperparams = dict([
123         ('batch_size', 500),
124         ('beam_size', 2),
125         ('dropout', 0.25),
126         ('attention_dropout', 0.0),
```

```
127         ('use_recategorization', False),
128         ('remove_wiki', False),
129         ('collapse_name_ops', False),
130         ('penman_linearization', True),
131         ('use_pointer_tokens', True),
132         ('raw_graph', False),
133         ('restore_name_ops', False),
134         ('return_all', False),
135         ('evaluation', True),
136         ('name', 'baseline+smart_init'),
137         ('model', 'facebook/bart-large'),
138         ('device', 'cpu' if args.cpu else 'cuda')
139     ])
140 
141     # La struttura dati che contiene i risultati
142     scores = dict()
143 
144     # I parametri vengono trascritti su file
145     with open(paths['metrics'], "w") as f_metrics:
146         for hp, value in hyperparams.items():
147             f_metrics.write(f'{hp}: {str(value)}\n')
148     state_dict_parsing = torch.load('modelli/AMR3.parsing.pt')
149     device = torch.device(hyperparams['device'])
150     model, tokenizer = instantiate_model_and_tokenizer(
151         hyperparams['model'],
152         dropout=hyperparams['dropout'],
153         attention_dropout=hyperparams['attention_dropout'],
154         penman_linearization=hyperparams['penman_linearization'],
155         use_pointer_tokens=hyperparams['use_pointer_tokens']
156     )
157     model = model.to(device)
158     model.load_state_dict(state_dict_parsing['model'])
159 
160 
161 # Iterazione su tutto il dataset passato in input, viene utilizzato\
162 # tqdm per ottenere una stima del tempo d'esecuzione
163 with tqdm(desc="labeling", total=len(dataset)) as bar:
164     for num, example in enumerate(dataset):
165         # Viene controllata la lunghezza del testo espresso in\
166         #token e confrontata con il numero massimo di token\
167         # elaborabili, se è minore non viene eseguita la computazione
```

```
168         if example[len_column] <= max_tokens:
169             scores[num] = []
170         else:
171             document = example[text_column]
172             summary = example[summary_column]
173             f_document = nltk.sent_tokenize(document)
174             f_summary = nltk.sent_tokenize(summary)
175 
176             if args.amr_smatch:
177                 # Creo rappresentazione AMR dei file in input
178                 if not os.path.exists(f'{paths["source_path"]}\
179                     /amr_source_{num}.txt')\
180                 or not os.path.exists(f'{paths["summary_path"]}\
181                     /amr_summary_{num}.txt'):
182                     prepare_amr(f_document, f_summary, paths,\
183                         hyperparams, model, tokenizer, device, num)
184                 scores[num] = calc_amr_smatch(\
185                     f'{paths["source_path"]}/amr_source_{num}.txt',\
186                         f'{paths["summary_path"]}/amr_summary_{num}.\
187                             txt')
188         bar.update(1)
189     return scores
```

La funzione viene richiamata passando come argomento i vari set di training, evaluation e test. Qualora si tratti di una delle metriche di similarità che sfrutta la rappresentazione AMR, controllo se i risultati sono già stati eseguiti, in caso negativo eseguo la computazione, altrimenti li leggo da file.

```
1  # Se la metrica richiesta non sfrutta la rappresentazione AMR oppure\
2  # i risultati non sono stati trascritti su file, elaboro la funzione\
3  # indicata precedentemente su training, evaluation e test set
4  if not (args.amr_smatch or args.amr_wwlk) or not\
5      os.path.isfile(f'{module_path}/{path_for_results}\
6          /train_scores.txt'):
7 
8      train_dataset_soft_scores = soft_labeling(args, train_dataset,\
9          text_column, summary_column, len_column, 'train')
10     eval_dataset_soft_scores = soft_labeling(args, eval_dataset,\
11         text_column, summary_column, len_column, 'eval')
12     test_dataset_soft_scores = soft_labeling(args, test_dataset,\
13         text_column, summary_column, len_column, 'test')
14 # Altrimenti li carico da file per processarli
```

```
else:
    import json
    train_dataset_soft_scores = json.load(open(f'{module_path}\
        /{path_for_results}/train_scores.txt'))
    eval_dataset_soft_scores = json.load(open(f'{module_path}\
        /{path_for_results}/eval_scores.txt'))
    test_dataset_soft_scores = json.load(open(f'{module_path}\
        /{path_for_results}/test_scores.txt'))

# Se l'args è amr e il file non è stato creato, scrivo su file\
# gli scores
if (args.amr_smatch or args.amr_wwlk) and not\
    os.path.isfile(f'{module_path}/{path_for_results}\
        /train_scores.txt'):
    import json
    with open(f'{module_path}/{path_for_results}/train_scores.txt',\
        'w') as f_train:
        f_train.write(json.dumps(train_dataset_soft_scores))
    with open(f'{module_path}/{path_for_results}/eval_scores.txt',\
        'w') as f_eval:
        f_eval.write(json.dumps(eval_dataset_soft_scores))
    with open(f'{module_path}/{path_for_results}/test_scores.txt',\
        'w') as f_test:
        f_test.write(json.dumps(test_dataset_soft_scores))
```

Qualora la metrica utilizzata sfrutti la rappresentazione AMR la computazione della prima parte termina qui, è necessario lanciare nuovamente il codice per elaborare i risultati ottenuti nel nuovo testo costruito sulla base del soft labeling. Se invece la metrica utilizzata non sfrutta la rappresentazione AMR, l'esecuzione procede e viene generato il testo riassunto da passare al modello.

4.6.2 Costruzione del testo basato su soft labeling

La seconda parte della computazione verte sul generare, a partire dai risultati del soft labeling, il testo da passare al modello.

```
# Se l'args non è amr o il file è già stato creato (situazione in cui\
# ho letto prima gli scores) procedo con la scrittura del nuovo dataset

# Questa funzione ricostruisce, dati i risultati di soft labeling,\
# i testi estratti dal dataset iniziale e il numero massimo di token\
# che il modello è in grado di elaborare, il nuovo documento,\
```

```
# mantenendo la struttura iniziale di separazione dei testi
def obtain_doc(result, example, text_column, max_tokens):
    document = example[text_column]
    num_docs = len(document.split(char_DS)[:-1])
    tokens = 2 + num_docs  # PRIMERA aggiunge due tokens
    sentences_sorted = []
    docs = nltk.sent_tokenize(document)

    num = []
    score = []
    doc_sep_count = 0
    for i in range(len(docs)):
        if char_DS not in docs[i]:
            num.append(i)
            score.append(result[i-doc_sep_count])
        else:
            doc_sep_count += 1
    results = [(n, s) for n, s in zip(num, score)]
    results.sort(reverse=True, key=lambda x: x[1])

    tokenizer = AutoTokenizer.from_pretrained("allenai/PRIMERA")

    for idx, _ in results:
        tokens += len(tokenizer(docs[idx], add_special_tokens=\
            False)["input_ids"])
        if tokens >= max_tokens:
            break
        else:
            sentences_sorted.append((idx, docs[idx]))
    sentences_sorted.sort(key=lambda x: x[0])
    for idx, sent in enumerate(docs):
        if char_DS in sent:
            for i, tuple in enumerate(sentences_sorted):
                pos = tuple[0]
                if pos > idx:
                    sentences_sorted.insert(i-1, (idx, char_DS))
                    break
    sentences_flat = [sent for _, sent in sentences_sorted]
    new_doc = " ".join(sentences_flat) + char_DS
    return new_doc

```

```
# Elabora il dataset iniziale costruendone uno nuovo basato sui\
# risultati del soft labeling
def get_new_dataset(dataset, dict_dataset, text_column,\
    summary_column):
    df = dataset.to_pandas()
    text_column_list = []
    summary_column_list = []
    for key in dict_dataset.keys():
        # La lunghezza del testo convertito in token minore della\
        # lunghezza massima elaborata dal modello viene gestita\
        # in questo modo, caricando nel nuovo dataset lo stesso\
        # testo del documento iniziale
        if len(dict_dataset[key]) == 0:
            text_column_list.append(dataset[key][text_column])
            summary_column_list.append(dataset[key][summary_column])
        else:
            new_doc = obtain_doc(dict_dataset[key], dataset[key],\
                text_column, max_tokens)
            text_column_list.append(new_doc)
            summary_column_list.append(dataset[key][summary_column])
    df[text_column] = text_column_list
    df[summary_column] = summary_column_list
    return Dataset.from_pandas(df)

# I risultati ottenuti vengono processati per generare un nuovo\
# testo, convertito poi a dataset per essere processato dal modello
new_train_dataset = get_new_dataset(train_dataset,\
    train_dataset_soft_scores, text_column, summary_column)
new_eval_dataset = get_new_dataset(eval_dataset,\
    eval_dataset_soft_scores, text_column, summary_column)
new_test_dataset = get_new_dataset(test_dataset,\
    test_dataset_soft_scores, text_column, summary_column)

new_dataset = DatasetDict({
    'train': new_train_dataset,
    'validation': new_eval_dataset,
    'test': new_test_dataset
})
new_dataset.save_to_disk(f'{module_path}/{path_for_results}/')
```

4.6.3 Esecuzione

Per eseguire il programma Python, è necessario integrare un modello per l'elaborazione delle strutture AMR. Sono stati predisposti script per l'installazione automatica delle librerie necessarie e per l'esecuzione del programma. Lo script per l'esecuzione della metrica smatch, ad esempio, è il seguente:

```
#!/bin/bash

pip install transformers
cd ./spring
pip install -r requirements.txt
pip install -e .
cd ..
python3 process_multilexsum.py --dataset multi_lexsum_long
    --amr_smatch --max_train_samples 100 --max_eval_samples 10
    --max_test_samples 100
pip install --upgrade transformers
python3 process_multilexsum.py --dataset multi_lexsum_long
    --amr_smatch --max_train_samples 100 --max_eval_samples 10
    --max_test_samples 100
```

Listato 4.1: script per l'esecuzione della metrica smatch.

La fase preliminare all'esecuzione è relativa alle installazioni delle librerie necessarie; successivamente, come già accennato, la computazione per le metriche che richiedono la rappresentazione AMR viene divisa in 2, alla fine della prima viene effettuato l'*update* della versione della libreria transformers e viene processato nuovamente il programma, che produce il testo da passare al modello.

4.7 Risultati

4.7.1 Metriche

Le metriche utilizzate nel soft labeling sono:

- **ROUGE-1**, per cui sono stati valutati i risultati di *precision* e *f-measure*;
- **ROUGE-2**, per cui sono stati valutati i risultati di *precision* e *f-measure*;
- **ROUGE-L**, per cui sono stati valutati i risultati di *precision* e *f-measure*;
- **BLEU**.

Sono state fatte prove con altre metriche, successivamente scartate:

- **BERTScore**, selezionato per includere una metrica di analisi semantica, successivamente scartato per 2 motivazioni:
  - Il calcolo delle similarità basate su questa metodologia è risultato fin troppo oneroso, e quindi non sempre applicabile a dinamiche reali;
  - La metrica risulta essere poco sensibile alla similarità tra frasi (come si evince dai test sperimentali in [22]), calcolata come una media dei punteggi massimi ottenuti token per token; questo problema rende meno preciso il soft labeling.
- **Smatch** e **WWLK**, selezionati per includere una metrica basata su rappresentazione semantica, anch'essi scartati per 2 motivazioni:
  - In fase preliminare estrarre la rappresentazione a grafi AMR è risultata molto onerosa in termini di risorse macchina e tempo, per cui non è sempre applicabile a dinamiche reali;
  - Considerando solamente le frasi rappresentate correttamente nella struttura a grafi per il calcolo dei punteggi, vengono escluse diverse frasi che andrebbero invece considerate: analizzando i problemi riscontrati nella creazione dei grafi, saltuariamente il modello non riesce ad elaborare simbologie e struttura delle frasi contenute nel dataset.

4.7.2 Tempistiche e prestazioni

I tempi medi di esecuzione del soft labeling di un documento di 5960 parole su un riassunto di 320 parole sono:

- **ROUGE-1 $\approx$ 1 secondo** sia per precision che per f-measure;
- **ROUGE-2 $\approx$ 1 secondo** sia per precision che per f-measure;
- **ROUGE-L $\approx$ 1.2 secondi** sia per precision che per f-measure;
- **BLEU $\approx$ 4.7 secondi**;
- **BERTScore $\approx$ 82.7 secondi** sfruttando l'elaborazione su GPU NVIDIA GEFORCE 3090[4];

[4] `https://www.nvidia.com/it-it/geforce/graphics-cards/30-series/rtx-3090-3090ti/`

- **Smatch:** $\approx$ **320 secondi** per la rappresentazione AMR sfruttando l'elaborazione su GPU NVIDIA GEFORCE 3090,
  $\approx$ **19.8 secondi** per il calcolo della metrica;

- **WWLK:** $\approx$ **320 secondi** per la rappresentazione AMR sfruttando l'elaborazione su GPU NVIDIA GEFORCE 3090,
  $\approx$ **4250 secondi** per il calcolo della metrica.

La frequenza media di corretta rappresentazione dai documenti ai grafi AMR è di $\approx 95\%$.

4.7.3 Esperimenti

Al fine di ottenere correlazioni più veritiere e risultati più precisi, sono state svolte prove differenti:

- Modificando il numero di documenti utilizzati in fase training;

- Variando il tipo di documenti impiegati in fase di training, passando al modello i documenti grezzi oppure una serie di frasi estratte dai documenti in input mediante metriche diverse di soft labeling;

- Variando il tipo di documenti impiegati in fase di test, passando al modello sia i documenti grezzi sia i testi ottenuti dall'estrazione delle frasi con soft labeling.

Tutti gli esperimenti utilizzano il modello addestrato su 10 epoche. Al termine di ogni epoca questo viene testato sul validation set contenente 10 documenti e viene registrato il risultato. Terminato l'addestramento, per le prove sul test set viene adottato il modello che ha ottenuto il risultato migliore sul validation set.

In dettaglio gli esperimenti svolti sono i seguenti:

1. PRIMERA addestrato su **100 documenti ottenuti dal soft labeling sul training set** e testato su **100 documenti ottenuti dal soft labeling sul test set**: ha l'obiettivo di individuare i **risultati nella miglior configurazione possibile**;

2. PRIMERA addestrato su **10 documenti ottenuti dal soft labeling sul training set** e testato su **100 documenti ottenuti dal soft labeling sul test set**: ha l'obiettivo di verificare la **correlazione tra il numero di documenti e la precisione ottenuta**;

3. PRIMERA addestrato su **tutto il training set** e testato su **100 documenti estratti dal test set**: ha l'obiettivo di individuare se il **training su un dataset più piccolo ma più informativo porti ad una precisione maggiore**;

4. PRIMERA addestrato su **tutto il training set** e testato su **100 documenti ottenuti dal soft labeling sul test set**: ha l'obiettivo di individuare **quanto il soft labeling incida a livello di test**;

5. PRIMERA addestrato **sia su 10 che su 100 documenti presi dal training set** e testato su **100 documenti presi dal test set**: ha l'obiettivo di dimostrare l'**effettiva efficacia del soft labeling**;

6. PRIMERA addestrato sia su **10 che su 100 documenti ottenuti dal soft labeling sul training set** e testato su **100 documenti estratti dal test set**: ha l'obiettivo di individuare **l'effetto di input di qualità in regime low-resources**.

Multi-LexSum	**LONG**	**SHORT**	**TINY**
Metrica	R1 / R2 / RL	R1 / R2 / RL	R1 / R2 / RL
BLEU	49.98/22.31/25.37	43.62/19.18/28.58	23.21/7.64/19.08
R1-F1	**51.76**/**23.58**/**26.12**	44.06/19.33/29.32	27.57/9.28/21.91
R1-P	50.30/22.72/25.60	40.97/16.96/26.51	27.64/10.25/22.74
R2-F1	51.16/23.45/25.82	**45.29**/**20.20**/**30.19**	27.61/10.18/21.58
R2-P	49.67/22.49/25.37	43.32/19.20/29.14	26.31/9.31/20.67
RL-F1	50.93/23.35/25.85	43.18/19.34/28.3	**29.01**/10.56/**23.95**
RL-P	47.76/21.29/24.68	43.90/19.14/29.15	28.48/**10.63**/23.79

Tabella 4.1: Risultati dell'esperimento 1: PRIMERA addestrato su 100 documenti ottenuti dal soft labeling sul training set e testato su 100 documenti ottenuti dal soft labeling sul test set. I risultati migliori sono in grassetto, i secondi sono sottolineati.

Multi-LexSum Metrica	**Long** R1 / R2 / RL	**Short** R1 / R2 / RL	**Tiny** R1 / R2 / RL
BLEU	43.30/16.49/20.75	37.39/13.17/22.69	23.59/6.75/17.68
R1-F1	43.93/17.29/21.32	32.33/9.35/18.95	**25.72**/7.57/**19.48**
R1-P	44.50/17.78/21.89	35.00/11.86/21.74	23.57/6.54/17.10
R2-F1	43.81/17.98/**22.33**	35.73/**14.40**/**23.64**	23.38/7.24/17.95
R2-P	**44.81**/**18.76**/22.30	35.67/12.67/22.22	24.64/**8.82**/18.77
RL-F1	43.60/17.72/21.87	**37.86**/13.19/23.21	21.43/6.01/17.35
RL-P	43.92/16.93/21.49	36.24/13.15/22.77	23.08/6.92/17.87

Tabella 4.2: Risultati dell'esperimento 2: PRIMERA addestrato su 10 documenti ottenuti dal soft labeling sul training set e testato su 100 documenti ottenuti dal soft labeling sul test set. I risultati migliori sono in grassetto, i secondi sono sottolineati.

Multi-LexSum	**Long** R1 / R2 / RL	**Short** R1 / R2 / RL	**Tiny** R1 / R2 / RL
No soft labeling	53.77/27.54/29.64	45.46/21.81/30.91	28.96/11.98/24.11

Tabella 4.3: Risultati dell'esperimento 3: PRIMERA addestrato su tutto il training set e testato su 100 documenti estratti dal test set.

Multi-LexSum Metrica	**Long** R1 / R2 / RL	**Short** R1 / R2 / RL	**Tiny** R1 / R2 / RL
BLEU	51.35/23.61/25.86	44.30/19.43/28.91	26.30/8.60/21.02
R1-F1	52.35/24.54/26.37	45.16/20.41/29.62	**30.57**/11.32/**24.20**
R1-P	51.27/23.92/26.55	45.44/20.50/30.00	30.32/**11.88**/24.16
R2-F1	**53.27**/**25.88**/**27.72**	**46.69**/**21.94**/**31.21**	28.38/11.32/23.73
R2-P	51.89/24.74/26.92	46.37/21.69/30.92	28.13/11.39/23.61
RL-F1	52.79/25.45/27.13	46.01/20.79/30.35	29.96/11.52/24.13
RL-P	51.75/24.64/26.85	46.06/21.39/30.84	28.07/10.66/22.61

Tabella 4.4: Risultati dell'esperimento 4: PRIMERA addestrato su tutto il training set e testato su 100 documenti ottenuti dal soft labeling sul test set. I risultati migliori sono in grassetto, i secondi sono sottolineati.

Multi-LexSum Documenti	**Long** R1 / R2 / RL	**Short** R1 / R2 / RL	**Tiny** R1 / R2 / RL
10	45.34/18.88/22.68	35.25/12.61/22.23	21.11/5.60/16.12
100	**48.81**/**22.25**/**24.93**	**42.30**/**17.96**/**28.22**	**23.94**/**7.04**/**19.32**

Tabella 4.5: Risultati dell'esperimento 5: PRIMERA addestrato sia su 10 che su 100 documenti presi dal training set e testato su 100 documenti presi dal test set. I risultati migliori sono in grassetto.

Multi-LexSum Metrica	**Long** R1 / R2 / RL	**Short** R1 / R2 / RL	**Tiny** R1 / R2 / RL
10 documenti			
BLEU	41.96/17.61/21.82	**36.26**/13.05/**22.87**	22.39/**6.57**/**17.68**
R1-F1	41.36/17.34/21.6	27.38/9.36/17.88	21.65/6.33/16.66
R1-P	43.05/18.99/22.66	33.94/11.88/21.49	**22.68**/5.59/16.29
R2-F1	41.66/17.56/22.05	32.33/12.86/21.69	21.33/5.47/16.48
R2-P	**43.30**/**19.11**/**22.81**	34.34/12.11/22.14	20.07/5.68/15.24
RL-F1	41.43/17.20/21.52	34.22/12.65/22.16	18.52/4.35/14.62
RL-P	42.85/18.29/22.55	35.34/**13.41**/22.71	20.39/4.95/15.45
100 documenti			
BLEU	45.96/**21.60**/25.12	**42.99**/**19.77**/**29.37**	23.49/7.44/19.67
R1-F1	46.02/21.13/25.02	42.29/18.64/28.65	24.43/8.48/20.14
R1-P	**46.45**/21.53/**25.17**	40.69/17.74/27.12	25.21/8.06/20.03
R2-F1	45.73/20.95/24.83	41.44/18.58/28.47	24.99/8.90/20.15
R2-P	45.96/21.31/25.05	41.04/17.69/28.04	22.43/7.30/17.62
RL-F1	45.46/21.02/25.02	42.12/19.40/28.69	26.15/**10.08**/**22.10**
RL-P	45.88/21.30/25.02	41.93/18.80/28.76	**26.57**/9.16/22.08

Tabella 4.6: Risultati dell'esperimento 6: PRIMERA addestrato sia su 10 che su 100 documenti ottenuti dal soft labeling sul training set e testato su 100 documenti estratti dal test set. I risultati migliori sono in grassetto, i secondi sono sottolineati.

4.8 Findings

Q1: Il training su un dataset più piccolo ma più informativo porta a generare riassunti più accurati? Basandosi sugli esperimenti 3 e 6, riportati in tabella 4.3 e 4.6, si è giunti alla conclusione che il modello perda di accuratezza di solo 2 punti di media per i dataset short e tiny, nonostante si addestri un modello con, rispettivamente, $\frac{1}{20}$ e $\frac{1}{10}$ di dati etichettati. Diversamente, il dataset long subisce la perdita maggiore, pari a circa 6 punti, utilizzando circa $\frac{1}{30}$ dei dati etichettati.

Q2: Quanto è efficace il metodo di soft labeling? Basandosi sugli esperimenti 5 e 6, riportati in tabella 4.5 e 4.6, si evince che nel confronto dei 100 documenti, l'applicazione del soft labeling faccia si che le metriche migliori ottengano risultati maggiori sul dataset short e tiny di 1 e 3 punti rispettivamente, registrando un nuovo stato dell'arte in condizioni low-resource con solo 100 documenti di training. Il dataset long presenta invece risultati comparabili.

Per quanto riguarda il confronto con 10 documenti, l'applicazione del soft labeling non porta risultati positivi: il dataset long perde circa 3 punti di media, lo short ne perde circa 2, mentre il tiny risulta comparabile.

Q3: Quanto può essere importante applicare il soft labeling a livello di test? Basandosi sugli esperimenti 4 e 3, riportati in tabella 4.4 e 4.3, si può notare come il dataset long peggiori i propri risultati, lo short li migliori invece di circa 1 punto, mentre il dataset tiny risulti essere comparabile. I risultati dimostrano l'importanza dell'utilizzo di dati di qualità in fase di training, mentre in fase di test il soft labeling non apporta un contributo significativo rilevabile.

Analizzando gli esperimenti 1, 2 e 6, riportati in tabella 4.1, 4.2 e 4.6, su 10 documenti il dataset long aumenta di circa 2 punti medi, così come lo short, mentre il tiny migliora di circa 3 punti; su 100 documenti il long registra un aumento di circa 4 punti medi, mentre lo short e il tiny di circa 2 punti.

Q4: Quale è il metodo migliore per soft labeling? Confrontando gli esperimenti 1 e 2, riportati in tabella 4.1 e 4.2, non si individua un metodo di spicco per il soft labeling; si può arrivare quindi alla conclusione che questa tecnica dipenda molto dalla natura del documento in input e del suo riassunto.

Conclusioni e Sviluppi Futuri

In questo elaborato sono state analizzate diverse metriche per effettuare soft labeling a supporto del problema di multi-document summarization in ambito legale. Alla luce delle prove effettuate, sono stati condivisi i risultati in termini di tempo medio di esecuzione e di precisione misurata in situazioni differenti, da cui sono state evinte diverse considerazioni. Non è emerso un metodo migliore per effettuare il soft labeling, i cui risultati dipendono dalla natura del documento e del riassunto fornito. Avendo però registrato un nuovo stato dell'arte in condizioni low-resource in cui viene applicato il soft labeling solo in fase di training, è auspicabile ampliare il lavoro di ricerca adottando la tecnica in ambiti diversi da quello legale e coinvolgendo metodi diversi di semantic search, più efficienti rispetto a quelli trattati nell'elaborato, basati su ricerche più approssimative (ad esempio Sentence-BERT).

Ringraziamenti

Il primo ringraziamento va al relatore di questo lavoro, il Prof. Gianluca Moro, che ha reso possibile tutto ciò e ha acceso il mio personale interesse nei confronti di questa splendida disciplina.

Molto importante è stato anche il ruolo ricoperto dal dott. Luca Ragazzi, molto disponibile insieme al relatore nel chiarire ogni mio dubbio e ideare insieme soluzioni a problemi sorti durante lo sviluppo del progetto.

Voglio ringraziare la mia famiglia e i miei amici per il supporto mostratomi prima e durante tutto il percorso che mi ha portato fino a questo punto, con un riguardo particolare nei confronti della mia ragazza che, oltre il supporto fornitomi, ha portato luce anche nei momenti più bui.

Dedico il mio lavoro e i miei risultati a tutti coloro che mi sono stati vicini e a chi purtroppo ha potuto partecipare solo all'inizio del tragitto.

Bibliografia

[1] Alan M. Turing. Computing machinery and intelligence. In Margaret A. Boden, editor, *The Philosophy of Artificial Intelligence*, Oxford readings in philosophy, pages 40–66. Oxford University Press, 1990.

[2] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[4] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, KyungHyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *CoRR*, abs/1506.07503, 2015.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[6] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Comput. Linguistics*, 19(1):61–74, 1993.

[7] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, 1958.

[8] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-based summarization of multiple documents: sentence extraction utility-based evaluation, and user studies. *CoRR*, cs.CL/0005020, 2000.

[9] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41(6):391–407, 1990.

[10] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

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

[12] Luciano Floridi and Massimo Chiriatti. GPT-3: its nature, scope, limits, and consequences. *Minds Mach.*, 30(4):681–694, 2020.

[13] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[14] Congbo Ma, Wei Emma Zhang, Mingyu Guo, Hu Wang, and Quan Z. Sheng. Multi-document summarization via deep learning techniques: A survey. *CoRR*, abs/2011.04843, 2020.

[15] Zejiang Shen, Kyle Lo, Lauren Yu, Nathan Dahlberg, Margo Schlanger, and Doug Downey. Multi-lexsum: Real-world summaries of civil rights lawsuits at multiple granularities. *CoRR*, abs/2206.10883, 2022.

[16] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In Stefanie Dipper, Maria Liakata, and Antonio Pareja-Lora, editors, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL 2013, August 8-9, 2013, Sofia, Bulgaria*, pages 178–186. The Association for Computer Linguistics, 2013.

[17] Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12564–12573. AAAI Press, 2021.

[18] Shu Cai and Kevin Knight. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 748–752. The Association for Computer Linguistics, 2013.

[19] Juri Opitz, Angel Daza, and Anette Frank. Weisfeiler-leman in the bamboo: Novel AMR graph metrics and a benchmark for AMR graph similarity. *Trans. Assoc. Comput. Linguistics*, 9:1425–1441, 2021.

[20] Wen Xiao, Iz Beltagy, Giuseppe Carenini, and Arman Cohan. PRIMERA: pyramid-based masked sentence pre-training for multi-document summarization. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5245–5263. Association for Computational Linguistics, 2022.

[21] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[22] Gianluca Moro and Luca Ragazzi. Semantic self-segmentation for abstractive summarization of long documents in low-resource regimes. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 11085–11093. AAAI Press, 2022.