Relazione Progetto di Sistemi Intelligenti, Machine Learning e Deep Learning

Autore: Maurizio Brini
Matricola:505195
Progetto Github: https://github.com/mauri5566/Progetto-SII-ML-DL/tree/main/Desktop/relazioniEsami

Nel presente rapporto sono descritte le attività eseguite per la realizzazione del progetto valevole per le materie Sistemi Intelligenti, Machine Learning e Deep Learning. Per le tre materie sono stati raggiunti rispettivamente i seguenti obiettivi:

- **Sistemi Intelligenti**: Sviluppo di un Web-Scraper per Recensioni di Ristoranti
- **Machine Learning**: Addestramento sul dataset ricavato e fine-tuning di modelli di machine learning per la Sentiment Analysis di recensioni
- **Deep Learning**: Studio di BERT e valutazione della sua applicazione nel task della Text Summarization.

Sviluppo Scraper per Recensioni di Ristoranti

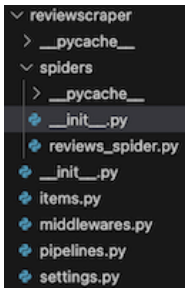
Nel contesto sempre crescente dell'analisi dei dati e dell'importanza del loro utilizzo, l'estrazione di informazioni da fonti online è diventata un elemento cruciale per molte applicazioni. Il **web scraping** (detto anche web harvesting o web data extraction) è una tecnica informatica di estrazione di dati da un sito web per mezzo di programmi software chiamati **scraper**. È realizzato con l'obiettivo di convertire il contenuto di un sito web in un formato strutturato come tabelle, *JSON*, *XML* ecc.

Il web scraping non potrebbe esistere senza il **web crawling**, il processo di navigazione del web alla ricerca delle informazioni di interesse. I **crawler** si muovono attraverso i siti web seguendo i link seguendo una logica ricorsiva definita dal programmatore.

Questo rapporto presenta un'analisi dettagliata del processo di sviluppo di uno scraper mirato all'estrazione di recensioni di ristoranti dalle tre principali piattaforme in questo ambito: *TripAdvisor*, *Quandoo* e *Yelp*. Il fine ultimo è stato quello di creare un dataset adatto all'addestramento di algoritmi di Machine Learning come Sentiment Analysis e Text Summarization. Implementato utilizzando il linguaggio di programmazione Python, il progetto ha comportato l'estrusione di un notevole volume di dati, per un totale di 376.273 recensioni.

Funzionamento di Scrapy

L'uso di **Scrapy**, un potente framework open-source per il web scraping e crawling, ha reso possibile la creazione di un'applicazione altamente efficiente ed efficace per raggiungere questo obiettivo. Un progetto in scrapy ha la seguente struttura:



Solo alcuni di questi file sono stati modificati per realizzare lo scraper:

- **items.py**: Definisce l'oggetto da estrarre e tutte le sue feature. Nel caso dell'oggetto *Review* sono state ritirate le seguenti caratteristiche:
  - ***quote***: Piccolo commento che riassume l'opinione dell'utente. Nella maggior parte dei casi non supera le 10 parole.
  - ***review***: Contenuto completo della recensione
  - ***restaurant_name***: Nome del ristorante recensito
  - ***rating***: Numero di stelle (da 1 a 5) assegnate dall'utente
  - ***address***: Indirizzo del ristorante
- **settings.py**: File di impostazioni che permettono di personalizzare il comportamento dello scraper. I parametri più comunemente utilizzati sono:
  - ***BOT_NAME***: È il nome del progetto che viene utilizzato nel comando per avviare l'esecuzione dello scraper
  - ***USER_AGENT***: Parametro utilizzato per identificare lo scraper ai server dei siti web di cui di vuole ritirare il contenuto. È importante impostarlo correttamente per non avere accesso negato.

    - ***CONCURRENT_REQUESTS***: Specifica il numero massimo di oggetti che Scrapy può processare in parallelo.
    - ***DOWNLOAD_DELAY***: È l'intervallo di tempo che separa i download delle pagine dello stesso sito. È a 0 di default ma può essere aumentato per non sovraccaricare il server.

- **reviews_spider.py**: Definisce la logica di crawling e di scraping del programma. È il file che ha costituito il focus principale della realizzazione del progetto.

Una componente fondamentale per il funzionamento di Scrapy sono gli oggetti ***Request***, utilizzati per seguire i link e spostarsi da una pagina all'altra. Il crawling della pagina è gestito da una funzione specificata nel parametro *callback*.

```
yield scrapy.Request(url, callback = self.parse)
```

Vengono effettuate delle richieste HTTP che generano un oggetto ***Response*** che diviene proprio della funzione di callback. Questo oggetto può essere utilizzato per estrarre informazioni dalla pagina corrente specificando le espressioni **xpath** o **css** che gli corrispondono.

```
def parse(self, response):

    item1 = response.xpath('espressione xpath')
    item2 = response.xpath('espressione css')

    yield item1,item2
```

Configurazione Iniziale

Dopo la corretta configurazione del file **items.py** per definire gli elementi da ritirare è stato necessario apportare delle modifiche ai valori di default del file **settings.py**.

Lo *USER_AGENT* di default può causare la segnalazione del sistema come bot e di conseguenza, il suo bloccaggio. Per evitare questi problemi è utile identificarsi con lo user agent della macchina in utilizzo. Per il progetto, eseguito con un MAC del 2022, è stato usato il seguente:

Il programma scarica contenuti in parallelo da multiple pagine di TripAdvisor, Yelp e Quandoo. Il numero massimo di richieste eseguibili in parallelo è impostato a 16 di default, questo valore si è rivelato troppo basso. Impostare *CONCURRENT_REQUESTS* a 50000 aggiungendo un *DOWNLOAD_DELAY* di 0,1 secondi ha evitato che lo scraping venisse bloccato.

Crawler

Sono state definite **tre diverse logiche di crawling**, dipendenti dall'url di partenza. I tre siti di recensioni presi in considerazione presentano diverse strutture e richiedono un trattamento diverso. Il programma fa riferimento a una funzione *start_requests* per avviare il crawling. In questa funzione, il crawler viene avviato in parallelo verso 48 url (specificati manualmente) di pagine contenenti liste di ristoranti, ognuna corrispondente a una delle 20 città più popolose in Italia.

TripAdvisor, il sito con la più larga utenza, viene esplorato a partire da 20 url. Per Yelp sono 17 le città con una quantità di recensioni ritenuta sufficiente mentre per Quandoo solo 11. Nel rapporto che segue sono descritte le scelte prese per la realizzazione del programma e le motivazioni che le hanno scaturite.

TripAdvisor

La funzione *start_requests* affida il compito di gestire i 20 url di partenza alla funzione *parseTripadvisorRestaurants*. Questa si occupa di individuare i ristoranti presenti nella pagina, trovare il link della loro pagina di dettaglio e seguirlo. La lista è composta dai 30 ristoranti migliori della città secondo gli utenti ma è presente anche un numero variabile di ristoranti sponsorizzati che sono solitamente 7. Di conseguenza, più di 700 pagine di ristoranti sono analizzate per ogni città.

La gestione della pagina di dettaglio è affidata alla funzione *parseTripadvisorRestaurant* che individua le 10 recensioni presenti e le altre informazioni di interesse, memorizzandole in una variabile *items*.

I ristoranti possono avere migliaia di recensioni distribuite su più pagine. Per questo *parseTripadvisorRestaurant* gestisce anche la **paginazione**. Dopo aver individuato l'url della pagina successiva (presente nel tasto *'next'*), gli invia una richiesta http per accedere al suo contenuto, contenuto la cui gestione è affidata alla medesima funzione. Le richieste http terminano quando il sistema si accorge della disabilitazione del tasto 'next', riscontrabile in un cambio di classe *html* del pulsante.

Lo scraping di TripAdvisor è risultato nell'estrazione di **355198** recensioni di cui:

- 251230 da 5 stelle
- 71013 da 4 stelle
- 18596 da 3 stelle
- 7893 da 2 stelle
- 6379 da 1 stella

C'è un chiaro squilibrio tra il numero di recensioni positive e il numero di recensioni negative. Essendo uno degli obiettivi del progetto quello di usare il dataset per la Sentiment Analysis delle recensioni, è stato ritenuto opportuno concentrare i successivi sforzi di sviluppo sul recupero di recensioni da 4 o meno stelle. Un **dataset equilibrato** è uno dei requisiti fondamentali per l'addestramento di un sistema adatto alla risoluzione di task come questo. Per questo motivo, il review scraping di Yelp e Quandoo ha seguito questa linea.

Quandoo

Gli 11 url di partenza sono gestiti dalla funzione *parseQuandooRestaurants* che individua gli url dei ristoranti presenti nella pagina. Quandoo offre la possibilità di ordinare le recensioni dalla peggiore alla migliore nella pagina di dettaglio del Ristorante. Visto che siamo interessati alle recensioni negative è conveniente usare questa funzionalità. L'idea è quella di continuare a estrarre recensioni fino a quando non si trova la prima recensione da 5 stelle.

Quando si filtrano le recensioni in questo modo, l'url della pagina di dettaglio viene modificato con l'aggiunta di un parametro *sortTypeId* a cui è assegnato il valore *lowest-rating*. Di conseguenza, *parseQuandooRestaurants* lo concatena agli url individuati. Successivamente porta lo scraper a seguire i link passando il controllo alla funzione *parseQuandooRestaurant*.

La gestione della **paginazione** sfrutta questo aspetto per interrompere l'invio di richieste alle pagine successive. Per ogni pagina viene recuperato il numero della pagina corrente (presente nell'url) e il rating dell'ultimo elemento di ogni pagina viene analizzato per verificare se è minore di 4. Se la condizione è verificata allora la variabile page_number viene incrementata e utilizzata per aggiornare l'url. In caso contrario il crawling dedicato al ristorante corrente termina.

Quandoo ha un sistema di punteggio che permette di assegnare anche 6 stelle a una recensione. Una volta recuperato il rating, questo viene riportato nella scala che va da 1 a 5 per concordare con i rating delle altre piattaforme.

È uno sito web principalmente usato da stranieri. L'obiettivo finale è quello di costruire un dataset che contenga esclusivamente recensioni in **lingua italiana** ma Quandoo non offre la possibilità di filtrarle in base al linguaggio in cui sono state scritte. Per risolvere il problema è stata utilizzata la libreria **langid** che fornisce uno strumento di riconoscimento del linguaggio, strumento che ha contribuito a stabilire una condizione necessaria per l'aggiunta della recensione al dataset finale.

Vista la piccola base di utenza italiana della piattaforma è stato implementato un sistema per gestire la paginazione anche per la pagine generali di partenza che contengono la lista dei ristoranti. Tuttavia il sistema si ferma alla quinta pagina vista la scarsità di recensioni presenti oltre quel punto.

Lo scraping da questa piattaforma ha portato all'estrazione di **9486 recensioni** di cui:

- 3383 da 5 stelle
- 3534 da 4 stelle
- 2183 da 3 stelle
- 241 da 2 stelle
- 145 da 1 stella

Yelp

La funzione *start_requests* affida la gestione delle 17 pagine di partenza di Yelp alla funzione *parseYelpRestaurants*. Quest'ultima individua gli url dei 10 ristoranti presenti nella pagina. Yelp offre la possibilità di filtrare le recensioni per punteggio. Quando questo viene effettuato, un parametro *rr* è aggiunto all'url. Seguendo la necessità di recensioni negative, la funzione concatena agli url questo parametro assegnandogli valori che vanno da 1 a 4, escludendo le pagine di dettaglio contenenti recensioni da 5 stelle. In modo analogo a Quandoo, anche la funzione di parsing generale gestisce la paginazione, questa volta non andando oltre la 24esima pagina, limite imposto da Yelp.

La funzione che naviga nelle pagine di dettaglio (filtrate con il punteggio) è *parseYelpRestaurant*. A causa di alcune limitazioni di Scrapy nell'estrazione di dati da siti web scritti in JavaScript non è stato possibile estrarre le recensioni dall'html della pagina corrente ma è stato recuperato un parametro *bizId* che costituisce una parte consistente dell'url dell'**API** della pagina corrente, un file *JSON* contenente i dati di interesse per l'attuale livello di filtraggio (numero di stelle).

Una volta composto l'url dell'*API*, la sua gestione è affidata alla funzione *parseYelpRestaurantApi*. Oltre ad accedere i campi del file JSON contenenti nome del ristorante, punteggio e recensione, verifica le condizioni per passare alla pagina successiva. Il campo *totalResults* indica il numero totale di recensioni del ristorante mentre *startResult* indica il numero di recensioni analizzate precedentemente. Facendo la differenza tra questi due valori è possibile verificare se la pagina corrente è anche l'ultima. In base all'esito di questa verifica si può decidere se passare all'API della pagina successiva.

Ritirare dati da Yelp si è rivelato problematico a causa di particolari policy del sito. Il server interrompe le risposte alle richieste http perchè riconosce il sistema di scraping come un bot non in linea con le lore regole di condivisione dei dati. Soltanto **428 recensioni** sono state ritirate con successo:

- 371 da 4 stelle
- 50 da 3 stelle
- 7 da 2 stelle

Machine Learning e Sentiment Analysis

Con la continua evoluzione del panorama digitale, le opinioni espresse online stanno diventando una fonte di informazione sempre più importante. La Sentiment Analysis gioca un ruolo cruciale nell'estrazione e nell'interpretazione di queste parole scritte. Questa tecnica consente di svelare gli aspetti emotivi nascosti nei dati testuali, fornendo un potente strumento per comprendere il pensiero delle utenze di una grande varietà di settori, dall'e-commerce ai social media nell'ambito della politica.

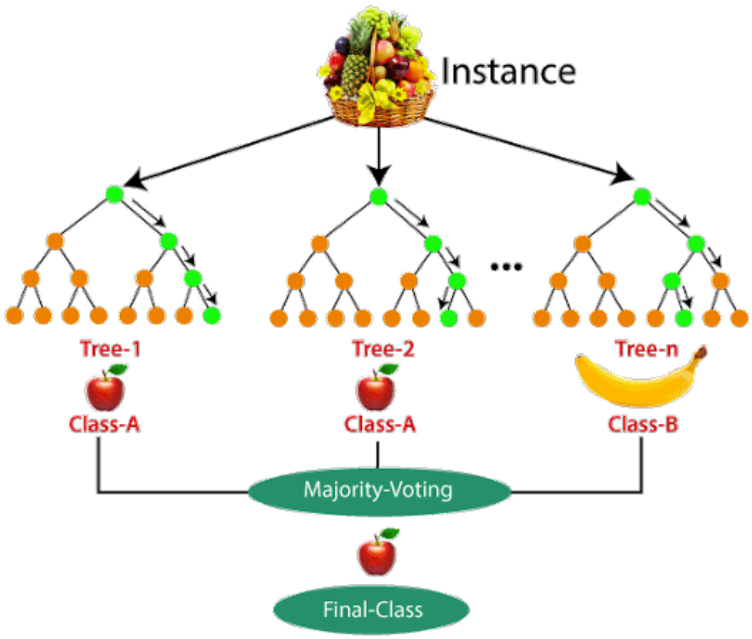
In questa relazione esploreremo nel dettaglio alcune delle tecniche per la realizzazione di questo tool. Inoltre ci concentreremo sull'importanza del loro utilizzo, analizzando i risultati di un testing effettuato su di esse e riportando i passi che si sono rivelati necessari per portarle al loro corretto funzionamento. Sono stati scelti due algoritmi: **Random Forest Classifier** e **Naive Bayes Classifier**

Random Forest Classifier

Il Random Forest è un algoritmo di machine learning la cui popolarità è stata agevolata dalla sua flessibilità nell'adattarsi sia a problemi di regressione che di classificazione. La forza principale di questo algoritmo sta nella sua capacità di gestire dataset complessi e mitigare overfitting. Nell'analisi seguente ci concentreremo sul task di classificazione

La caratteristica che distingue l'RFC dalle altre metodologie di machine learning è quella del **bagging**: viene combinato l'output di più strutture ad albero decisionali per raggiungere un unico risultato. Andiamo a vedere nel dettaglio quali sono gli step per realizzare questa tecnica:

- **Bootstrapping**: N record e m feature sono selezionati dal dataset originale
- **Training Indipendente**: Ogni albero decisionale viene addestrato indipendentemente sul suo campione di bootstrap corrispondente.
- **Majority Voting**: L'output finale è determinato combinando i risultati di tutti i modelli. Per i problemi di classificazione, è selezionato il risultato più comunemente predetto. Questo step è anche detto **aggregazione**.



Le strutture ad albero decisionali corrono il rischio di overfitting in quanto tendono ad adattarsi strettamente a tutti i campioni all'interno dei dati di addestramento. Tuttavia, quando esiste un numero consistente di strutture ad albero decisionali in un Random Forest, il classificatore non si adatta eccessivamente al modello poiché la diversità di alberi decisionali indipendenti riduce la varianza complessiva e l'errore di previsione.

Dal momento che gli algoritmi Random Forest possono gestire dataset di grandi dimensioni, possono fornire previsioni più accurate, ma possono essere lenti a elaborare i dati perché calcolano risultati per ogni singola struttura ad albero decisionale.

Naive Bayes Classifier

Il classificatore Naive Bayes è un algoritmo che sfrutta il **teorema di Bayes**. Il teorema di Bayes calcola la probabilità di un evento A condizionata a un altro evento E.

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Nell'ambito della classificazione questo teorema è utile per calcolare la probabilità che, dato un'elemento con un insieme di attributi X, questo appartenga alla classe y. Essendo X un insieme di feature, il teorema può essere espanso nel modo seguente.

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Vogliamo assegnare una sola classe a ogni elemento quindi viene scelta quella che ha la maggiore probabilità condizionata all'insieme di attributi X preso in considerazione.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Il motivo per cui questo classificatore è detto *'Naive'* (trad. Ingenuo) è perchè si basa sull'assunzione per cui tutte le feature sono indipendenti tra loro. Un frutto può essere classificato nella classe "mele" se è di colore rosso, ha un diametro di circa 10 cm e ha una forma rotonda. Sono tre caratteristiche differenti. Un algoritmo di Naive Bayes classifier considera ognuna di queste caratteristiche come un contributo indipendente alla probabilità complessiva che il frutto sia una mela. Nonostante questa semplificazione l'NB classifier rimane molto popolare grazie all' efficienza e precisione nelle sue applicazioni pratiche nel mondo reale.

Testing

Il presente rapporto illustra il lavoro svolto in **Python** per svolgere la Sentiment Analysis sulle recensioni di ristoranti scritte dagli utenti Italiani di *Tripadvisor*, *Quandoo* e *Yelp*. Dopo una fase iniziale che ha congiunto un'analisi del dataset con un suo text processing, un **Random Forest Classifier** e

un **Multinomial Naive Bayes Classifier** sono stati addestrati sui dati disponibili e ne sono state valutate le prestazioni.

Text Processing

La libreria **Pandas** si è rivelata fondamentale lungo tutto l'arco del progetto. Ha infatti permesso di memorizzare il dataset all'interno di un *dataframe*, un tipo di struttura di dati bi-dimensionale, una sorta di tabella con colonne e righe indicizzate. Questo strumento ha facilitato notevolmente la visualizzazione, l'accesso e la manipolazione dei dati.

Una volta acquisito il dataset, il suo contenuto è stato pulito per essere analizzato e preparato all'addestramento dei classificatori. Tutte le modifiche applicate sono state mirate a favorire l'apprendimento dei classificatori il cui compito è quello di assegnare valenze positive o negative alle parole. Utilizzando le espressioni regulari e il modulo '*re*' è stato possibile effettuare:

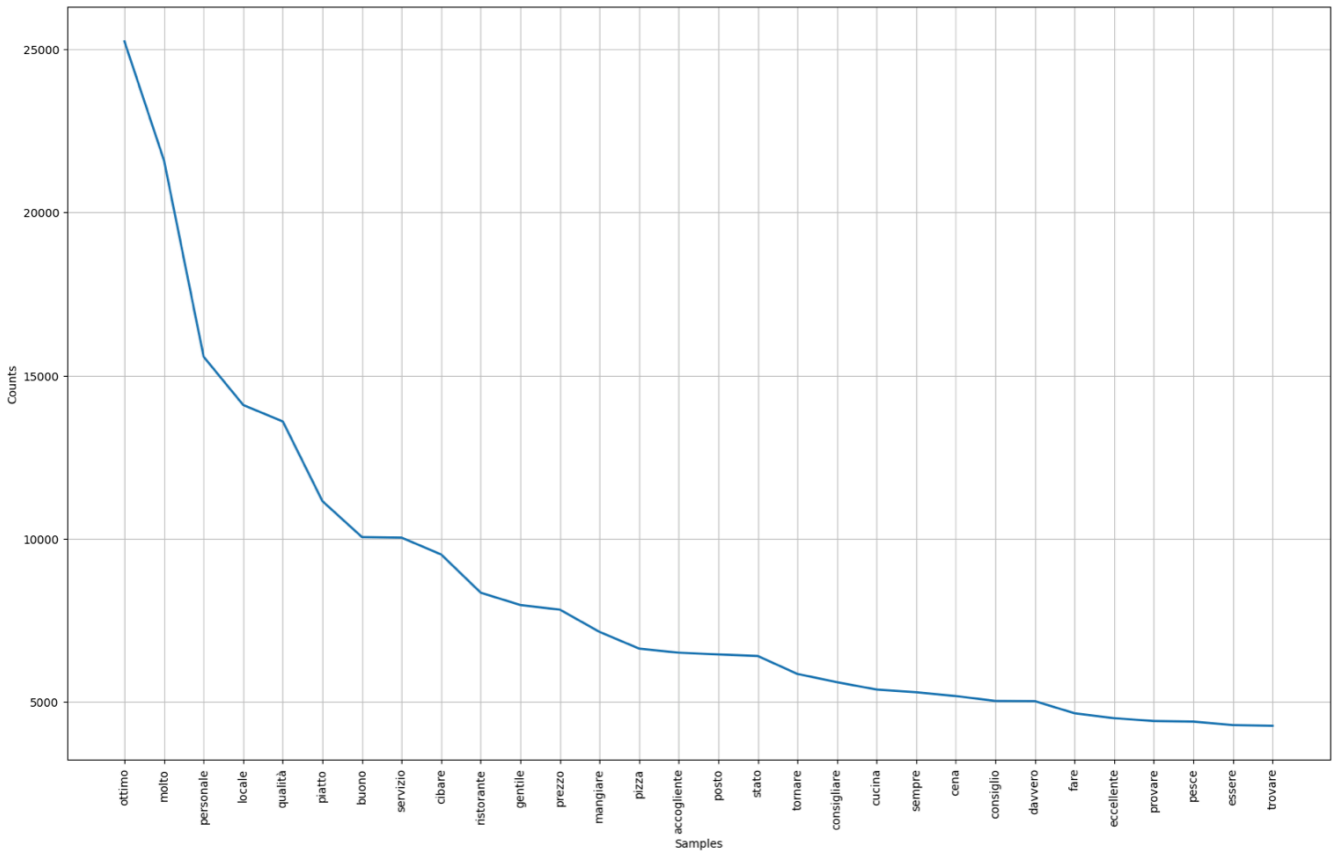
- Rimozione della **punteggiatura**, sostituita con uno spazio
- Rimozione di **spazi consecutivi** (errori di battitura dell'utente), sostituiti con un singolo spazio
- **Conversione in minuscolo** di tutte le lettere maiuscole. Il classificatore considera diversa una parola scritta in maiuscolo dalla sua corrispondente in minuscolo. Vogliamo evitare questa inconvenienza.

Sfruttando il modulo **nltk** sono state rimosse le stopword italiane. Questa modifica è necessaria perchè le stopword sono le parole più frequenti all'interno del testo e, se non rimosse, rischiano di ostacolare l'apprendimento dei modelli di machine learning.

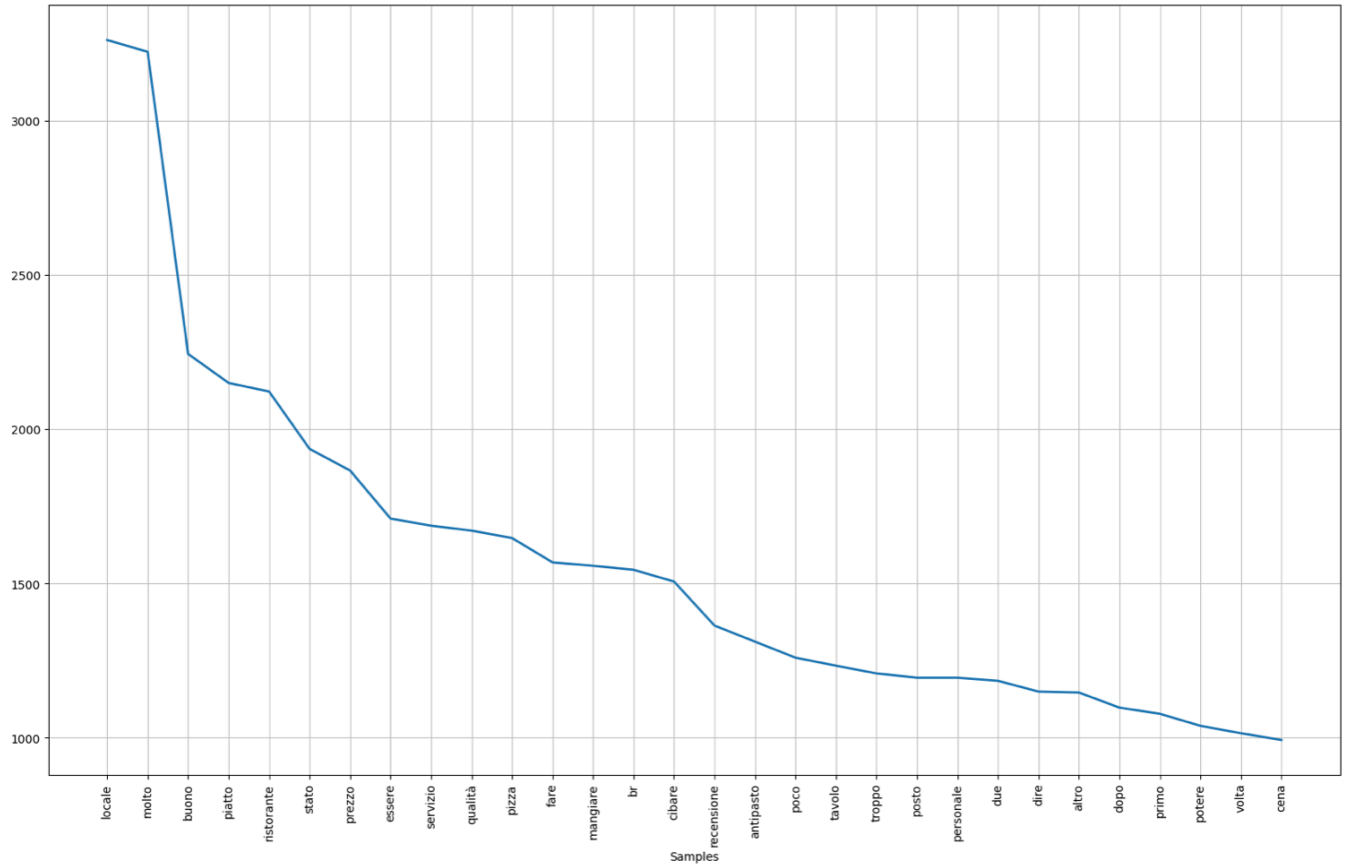
È stata poi applicata la **lemmatizzazione** delle parole restanti usando **simplemma**, un modulo che supporta questo processo per la lingua italiana. La lemmatizzazione consiste nella riduzione delle forme flesse delle parole alle loro forme canoniche. Vogliamo, ad esempio, che le occorrenze delle parole come 'accoglienza' e 'accolti' contribuiscano al conteggio totale della parola 'accoglienza'.

Analisi della frequenza

Il classificatore riceve in input un vettore che contiene la frequenza (conteggio) di ogni parola che appare nella recensione. È quindi utile svolgere un analisi della **distribuzione** delle parole all'interno delle recensioni. Gli utenti hanno assegnato un punteggio da 1 a 5 ai ristoranti valutati. Di seguito la distribuzione di frequenza delle parole per le recensioni da 5 stelle:



Per le recensioni da 1 stella la distribuzione è diversa:



Addestramento

I classificatori sono stati addestrati con l'obiettivo di stabilire se una recensione fosse positiva (*rating >= 4*) o negativa (*rating <= 3*). Vista la prevalenza di recensioni positive nel dataset, queste sono state in gran parte rimosse per favorire un addestramento equilibrato. Ne è risultato un dataset con 80100 record equamente divisi tra recensioni positive e negative.

La libreria **scikit-learn** è stata la colonna portante di questa fase del progetto. Fornisce gli strumenti necessari per implementare modelli di machine learning, addestrandoli e ricavando predizioni da essi.

Prima di procedere all'addestramento , le recensioni sono state sottoposte a **count vectorization**, un processo con cui un insieme di documenti di testo viene convertito in una matrice di conteggio dei token presenti in essi. Quando si usa questa tipo di vettorizzazione, l'assegnazione di una recensione a una classe piuttosto che a un altra dipende dalla frequenza con cui alcune parole occorrono all'interno di essa. In questo modo il dataset, costituito da recensioni vettorizzate e il loro corrispettivo label positivo/negativo è diviso in training e test set con una proporzione 75/25.

Random Forest

Il classificatore Random Forest fornito da scikit-learn può essere configurato in base a vari parametri. Sono stati presi in considerazione:

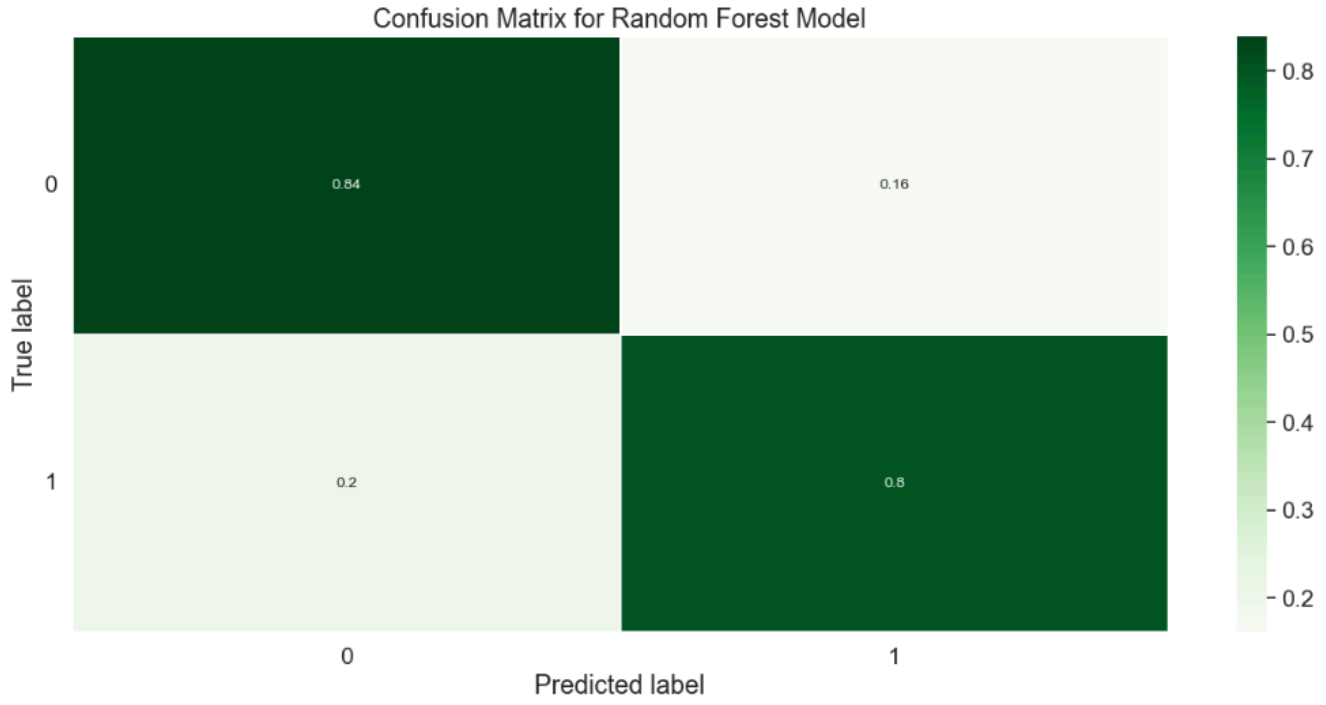
- ***bootstrap***: Valore booleano che indica se i campioni di bootstrap saranno utilizzati nell'addestramento degli alberi decisionali. Se *False*, l'intero dataset è utilizzato per ogni albero
- ***max_depth***: Indica la profondità massima che possono raggiungere gli alberi, ovvero il percorso più lungo tra nodo radice e foglia.
- ***minSamplesLeaf***: Questo parametro indica il numero minimo di osservazioni che dovrebbero essere presenti nei nodi foglia in seguito allo splitting di un nodo.
- ***minSamplesSplit***: Indica il numero minimo di osservazioni che devono essere presenti in un nodo per consentire il suo splitting (generazione dei suoi figli).
- ***n_estimators***: Il numero di alberi decisionali
- ***criterion***: Definisce la metrica di scelta dello split migliore. Può essere 'gini', 'entropy' e 'log loss'.

È stato testato un classificatore iniziale con 10 alberi ed 'entropy' come criterio di scelta dei nodi. L'*accuracy* che ne è risultata è stata di **0.784**, un buon risultato di partenza. Si è quindi proceduti con una fase di **fine-tuning** degli iperparametri. È stata effettuata una **Randomized Grid Search** sui possibili valori dei parametri sopra-menzionati. 100 differenti combinazioni di parametri sono state valutate in base alla media della loro accuracy su 3 addestramenti separati. I risultati hanno portato a una migliore consapevolezza sul comportamento del classificatore al variare dei parametri e a restringere lo spazio di ricerca del fine-tuning. Un alto numero di *estimators*, profondità illimitata e bootstrap sampling attivo sono tutte caratteristiche che si sono rivelate utili per aumentare la percentuale di predizioni corrette. I parametri restanti sono rimasti incerti. Tra le 100 iterazioni di addestramento, la migliore ha portato a un accuracy score di **0.815**

Una volta definiti i possibili parametri ottimali è stata effettuata una **grid search esaustiva**. 36 possibili combinazioni sono state valutate in modo analogo. I parametri migliori si sono rivelati essere:

- ***bootstrap***: True
- ***max_depth***: None
- ***minSamplesLeaf***: 2
- ***minSamplesSplit***: 2
- ***n_estimators***: 1000
- ***criterion***: entropy

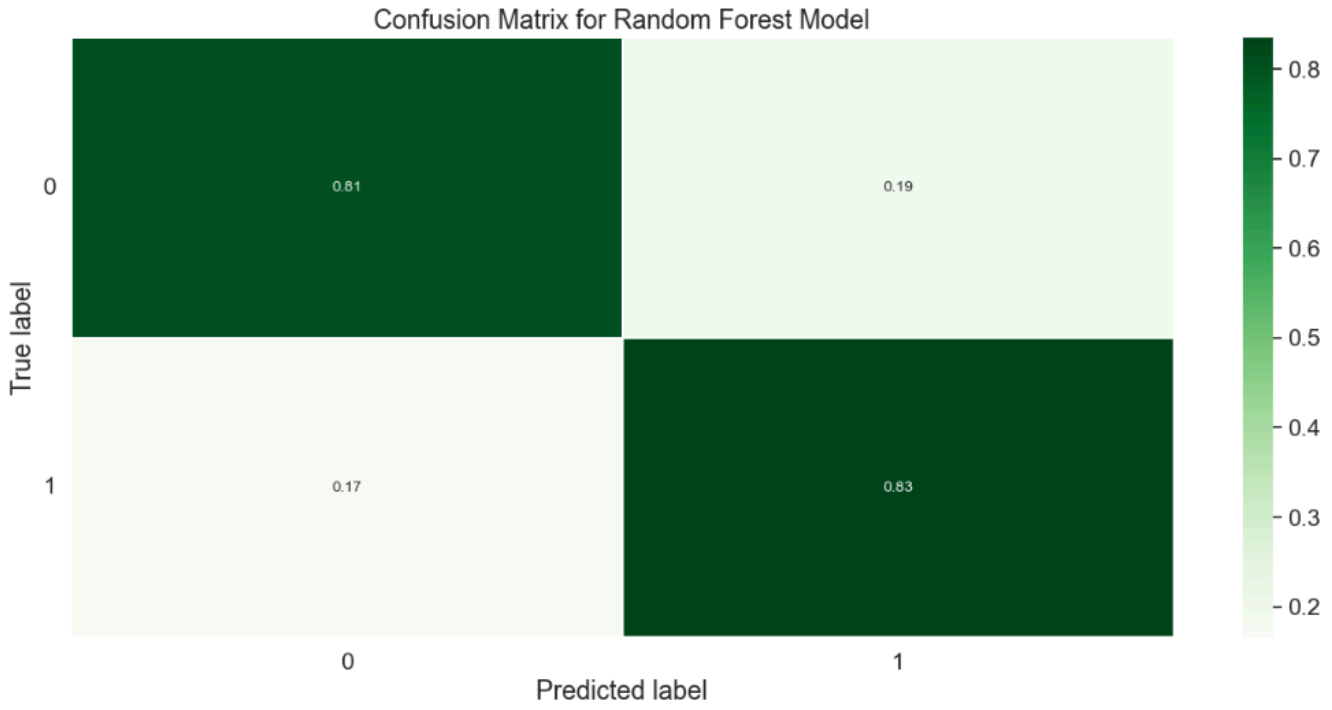
Dopo un tempo di addestramento di 7 minuti e 48 secondi, la frazione di predizioni corrette su quelle totali è stata di **0,818**. Come mostra la *confusion matrix*, il classificatore ha avuto meno difficoltà a riconoscere le recensioni negative raggiungendo una *recall* di 0.84 rispetto alla 0.80 di quelle positive



Naive Bayes

Tra i vari tipi di classificatori Naive Bayes messi a disposizione da *scikit-learn* è stato scelto quello **Multinomiale**, adatto a presenza di feature discrete come quella del word count, utilizzata in questo progetto per misurare il numero di occorrenze delle parole nelle recensioni. Tempi di addestramento notevolmente diminuiti rispetto al Random Forest e un unico parametro da ottimizzare hanno reso l'implementazione molto più immediata. Il parametro ***alpha*** indica l'intensità (da 0 a 1) dello *smoothing di Laplace*.

In seguito a una **grid search esaustiva**, l'assegnazione migliore per *alpha* si è rivelata essere di 0,0001, portando a un' *accuracy* di **0.821**, migliore rispetto al random forest



Approcci di Deep Learning per la Text Summarization e Valutazione con BERT

Nell'era digitale, le informazioni sono presenti principalmente sul web e la mole di dati esistente è in continua crescita. Una tale quantità suscita la necessità di sviluppare algoritmi che possano accelerare il processo di ricerca di informazione e ridurre i tempi di lettura. I riassunti svolgono un ruolo fondamentale in molte sfere della vita, sia in ambito professionale che personale, grazie alla loro capacità di condensare informazioni complesse in un formato più breve e facilmente comprensibile.

La **Text Summarization** è un task di Machine Learning, nell'ambito del **Natural Language Processing**, che consiste nel creare un riassunto dato un testo di input. L'obiettivo è quello di replicare il processo mentale umano per ottenerne una versione abbreviata. Esistono due principali approcci per l'automatizzazione di questo compito:

- **Approccio Estrattivo:** Questo approccio consiste nell'estrazione di frasi chiave dal testo originale e nella loro combinazione per creare un riassunto completo. Le frasi estratte non sono modificate.
- **Approccio Astrattivo:** Il metodo astrattivo crea nuove frasi in modo da evitare le inconsistenze grammaticali che quello estrattivo può generare. Affidandosi a tecniche di deep learning riesce a riprodurre più fedelmente il comportamento umano.

Esistono anche approcci misti ma tradizionalmente sono sempre stati utilizzati quelli estrattivi, mantenuti popolari dalla difficoltà di implementazione di quelli astrattivi, nati più recentemente con l'ascesa delle reti neurali e del Deep Learning

Approccio Estrattivo

L'approccio estrattivo riesce a mantenere il significato originale del testo con particolare efficacia. Questo metodo funziona bene quando l'input è ben strutturato. In caso contrario possono sorgere problemi di coerenza nella sintesi. Inoltre, a volte, le informazioni salienti sono distribuite su più frasi e l'approccio estrattivo non può catturare questo aspetto.

Esistono varie tecniche per realizzarlo ma tutte condividono la stessa struttura di base:

- Costruzione di una **rappresentazione intermedia** del testo in input
- **Valutazione delle frasi** basata sulla rappresentazione intermedia e su una funzione di scoring
- **Scelta delle frasi** in base ai punteggi assegnati per comporre l'output

Il primo task merita un approfondimento adeguato. Le due principali categorie di rappresentazioni intermedie sono la **topic representation** e l'**indicator representation**

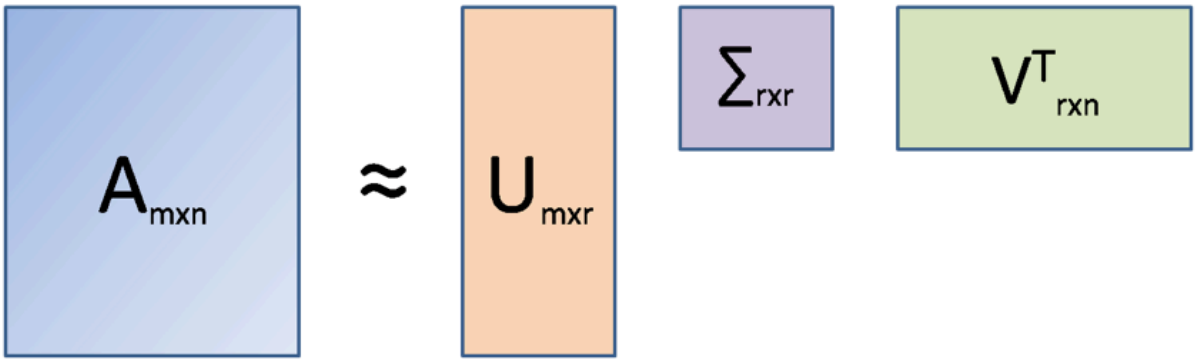
Topic Representation

Questo tipo di rappresentazione mira a catturare le frasi rilevanti basandosi sul tema che trattano. I documenti di testo sono solitamente scritti in modo da affrontare diversi argomenti, uno dopo l'altro, in maniera organizzata. È intuitivo pensare che anche le loro sintesi debbano fare riferimento a tali argomenti.

Tra le tecniche più utilizzate c'è quella del riconoscimento delle **topic words**, parole che risultano esplicative per il tema che trattano. Viene tenuto conto della distribuzione di queste parole per generare una rappresentazione adatta delle frasi. Più ne contengono, più le frasi risulteranno salienti.

Gli approcci **frequency-driven** sono estremamente popolari grazie alla loro semplicità e immediatezza. Tecniche come **word probability** e **tf-idf** definiscono una proporzionalità inversa tra frequenza di riscontro di una parola con la sua importanza.

La **Latent Semantic Analysis (LSA)** offre una soluzione al problema dell'identificazione dei temi trattati da una frase. Dati *m* termini e *n* frasi presenti in un documento, viene computata una matrice A *mxn* contenenti i valori *tf-idf* di ogni coppia termine-frase. La matrice è scomposta in tre sotto-matrici usando la **singular value decomposition (SVD)**.



Si ottengono due matrici ortogonali U e V. Le colonne della prima e le righe della seconda sono vettori rispettivamente corrispondenti ai termini e alle frasi pesati relativamente allo spazio dei temi. Σ è una matrice diagonale contenente i pesi di ogni argomento. Utilizzando queste informazioni, viene scelta una frase per ogni argomento trattato dal documento.

Indicator Representation

Gli approcci di questo tipo descrivono ogni frase con una serie di **features** (indicatori) salienti come la lunghezza della frase e posizione nel documento. Frasi molto lunghe o molto corte sono solitamente escluse dal riassunto mentre le prime e le ultime frasi dei primi e ultimi paragrafi hanno maggiori probabilità di essere importanti per il significato del documento.

Una grande varietà di caratteristiche sono prese in considerazione: Se la frase contiene parole presenti nel titolo, acronimi o nomi propri sarà più frequentemente selezionata. Il criterio più comunemente utilizzato è quello delle **keyword**

Le parole chiave sono individuate usando la metrica **tf x idf** ma possono essere riconosciute anche grazie a diversi metodi di keyword extraction. Le frasi contenenti le cosiddette *content words* sono ottime candidate per essere incluse nel riassunto finale.

Determinare l'importanza delle frasi basandosi sulle feature raccolte è un compito che viene spesso assegnato a metodi basati su **grafi** o a tecniche di **machine learning**.

Nel primo caso, i documenti sono rappresentati come grafi connessi in cui le frasi simili (*cosine-similarity*) sono connesse tra loro. La costruzione di questa struttura porta a due risultati: partizionando il grafo in sotto-grafi è possibile individuare gli argomenti trattati dal documento. Dall'altra parte, le frasi che hanno molte connessioni nella stessa partizione sono evidentemente molto descrittive dell'argomento che trattano.

Le tecniche ML modellano la summarization come un problema di classificazione. Un **classificatore Naive-Bayes** può classificare le frasi come riassuntive o non riassuntive in base alle feature che posseggono.

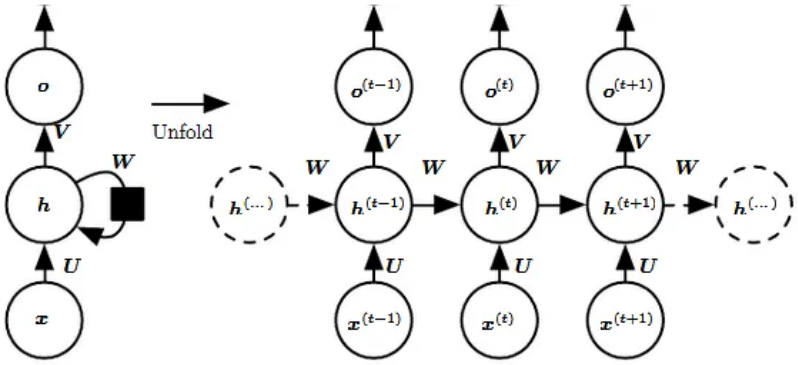
Approccio Astrattivo

Invece di selezionare delle frasi dal documento in input, questo approccio produce una parafrasi del suo contenuto principale mettendo in atto processi di comprensione del contesto, processi che sarebbero impossibili da realizzare senza l'utilizzo di **reti neurali**. Pur essendo applicabili anche all'approccio estrattivo, queste tecniche si sposano alla perfezione con la necessità di svolgere compiti con modalità di ragionamento simili a quelle umane. Nella sezione successiva saranno descritte le principali architetture utilizzate per la realizzazione di sintesi astrattive.

Recurrent Neural Networks

Una delle colonne portanti del Natural Language Processing è quella delle **reti neurali ricorrenti**, una classe di reti neurali adatta ad input sequenziali denominati **serie temporali**. Una *time series* consiste in una serie di misurazioni indicizzate con un ordine temporale. Le *RNN* sono dette ricorrenti perchè svolgono lo stesso task per ogni step temporale. L'output restituito dipende anche dalle computazioni precedenti.

Le RNN sono in grado di accettare sequenze in **input con lunghezza variabile**. Il documento da riassumere è trattato come una sequenza di parole, ognuna delle quali è assegnata a uno step temporale della time series.



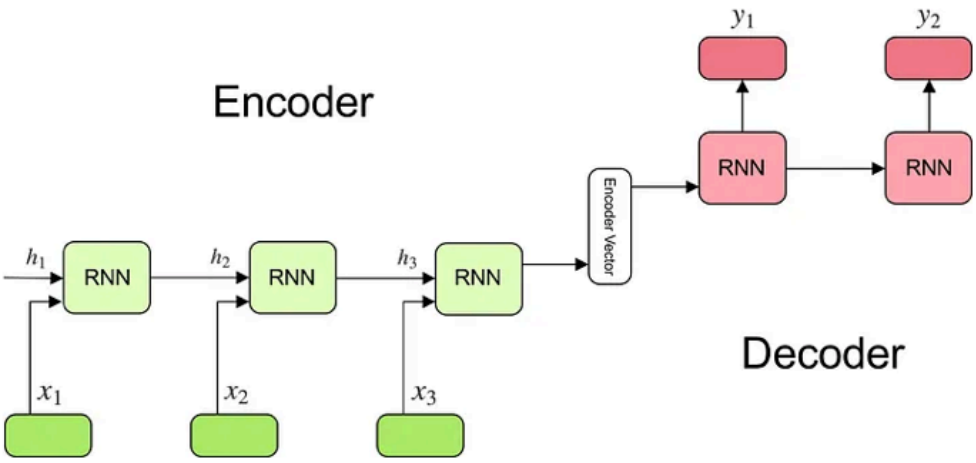
Ogni input (parola) *x(t)* è processato da una cella che produce un output *o(t)*, conservato nell'output finale e passato anche allo step temporale successivo. Indichiamo lo stato di una cella all'istante t con la notazione *h(t)*.

L'**hidden state** svolge il fondamentale ruolo della **memoria**. È importante tenere in considerazione i time step precedenti quando si processa quello attuale per comprendere il contesto in cui ricade l'input preso in considersazione. L'hidden state è aggiornato a ogni time step in base all'input corrente e all'hidden state precedente, elementi pesati rispettivamente con matrici dei pesi U e W.

```
h(t) = f(U x(t) + W h(t-1))
```

Il task della text summarization, oltre a comportare una sequenza in input (documento da riassumere), richiede anche la generazione di una sequenza in output (riassunto). Queste due sequenze saranno di diverse lunghezze. Da questa necessità, scaturita anche da molti altri task di NLP, nascono le reti **sequence-to-sequence**. Ecco come funzionano:

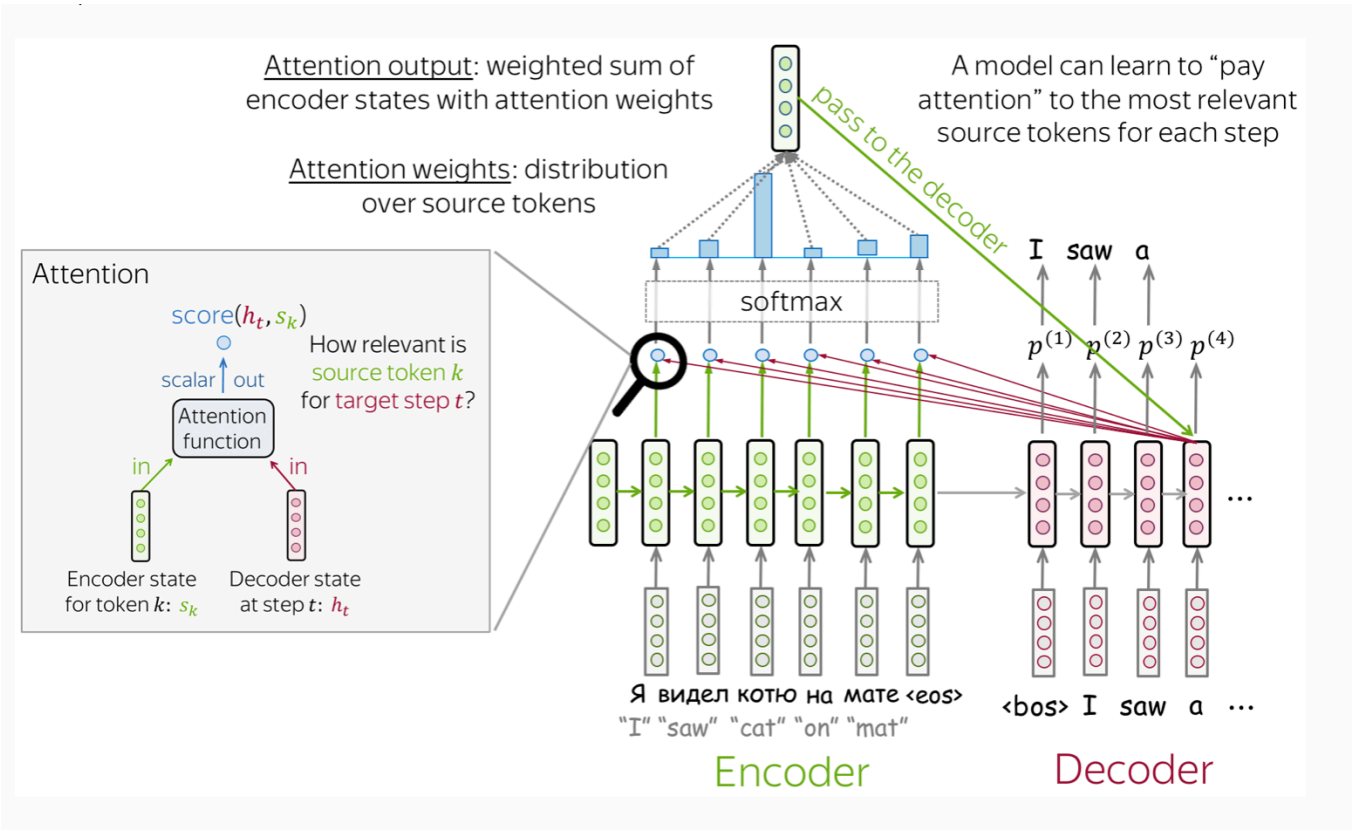
- Un layer RNN funziona da **encoder**, processa la sequenza in input generando una sua rappresentazione codificata detta **context vector**, ovvero il suo stato finale. Tutti i restanti output sono scartati.
- Il **decoder** è un altro layer RNN che acquisisce il context vector generato dall'encoder e lo utilizza come stato iniziale.Come per l'encoder, l'hidden state viene aggiornato ma tutti gli output sono conservati, in modo da formare una sequenza.



Nella fase di **training** si utilizza la tecnica del **backpropagation through time**: Una volta generati i risultati, questi vengono comparati con il *target output* per calcolare la metrica di *loss*. Il gradiente risultante è propagato in direzione inversa in modo da aggiornare i parametri della rete (che sono condivisi tra le celle).

Un'importante aggiunta a questa architettura è quella del **meccanismo di attenzione**. Quando le sequenze in input assumono grandezze maggiori diventa più difficile sintetizzarle in unica rappresentazione ridotta. Per risolvere questo problema si dedica più spazio o attenzione alle sezioni ritenute più importanti.

L'idea è quella di utilizzare un diverso context vector per ogni step di decodifica. Questo vettore è ottenuto da un'opportuna combinazione degli hidden state ottenuti in fase di codifica. In altre parole, il decodificatore può porre maggiore attenzione su una parte dell'input piuttosto che su un'altra a seconda della fase di decodifica in cui si trova. In questo modo può avere accesso ad informazioni distribuite su più vettori piuttosto che su uno solo, dove sono eccessivamente compresse.



Per ogni step del decoder un layer di attention:

- Riceve tutti gli hidden state dell'encoder e lo stato corrente del decoder *h(t)*.
- Calcola gli **attention scores** *s(k)* di ogni stato *k* in base alla rilevanza che ha per lo stato corrente del decoder. Questo viene effettuato applicando una funzione *score(h(k),h(t))*
- Calcola gli **attention weights**, distribuzioni di probabilità ottenute con l'applicazione di una softmax
- Calcola l'**attention output** che altro non è che una somma pesata degli score di attenzione

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$$

"source context for decoder step t"

Tranformer

I **Transformer** sono un'altra architettura introdotta nel 2017 strettamente basata sui meccanismi di attenzione. Ha raggiunto lo stato dell'arte per molti task Di Natural Language Processing inclusa la text summarization. Al contrario dei modelli precedenti, basati su ricorrenza (o convoluzioni), i transformer fanno affidamento esclusivamente al concetto di **attenzione**.

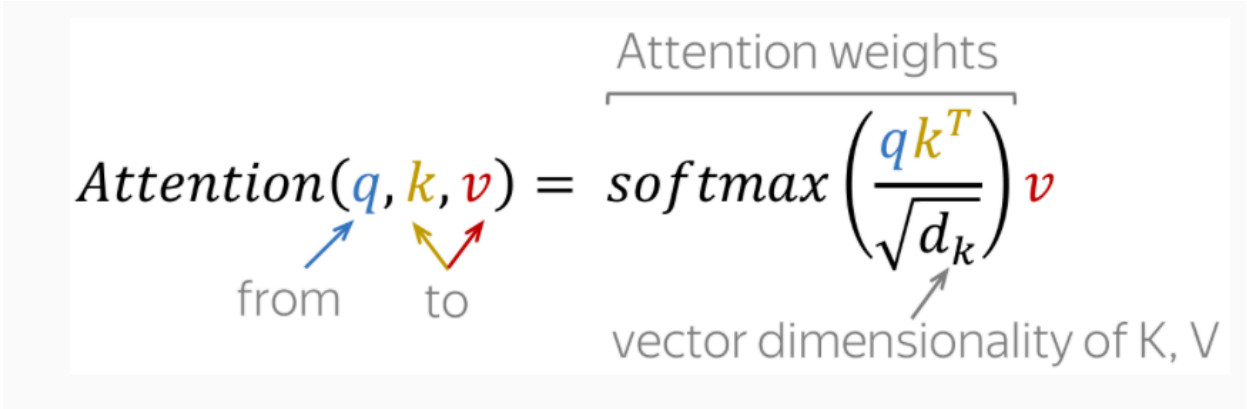
Necessita una grande quantità di dati e di potenza computazionale per essere addestrato quindi, al contrario delle RNN, non è sempre una scelta disponibile. Tuttavia presenta numerosi vantaggi rispetto ad esse. Uno dei motivi principali per cui utilizzare un transformer al posto di una RNN è la sua capacità di comprendere il linguaggio. Mentre le reti ricorrenti leggono una frase un passo alla volta, i Transformer interagiscono contemporaneamente con tutte le parole della sequenza in input al contrario delle RNN che devono aspettare di leggere tutta la frase prima di comprendere il significato delle parole intermedie.

Self-Attention

Tutti gli input tokens interagiscono tra loro in parallelo per generare una codifica. Questo meccanismo introduce il concetto di **self-attention**, diverso dall'attention perchè pone enfasi su operazioni tra stati **nello stesso layer**. Mentre nei meccanismi di attenzione gli stati dell'encoder sono combinati con lo stato del decoder corrente con la self-attention ogni stato dell'encoder interagisce con tutti gli altri stati dell'encoder (in parallelo). Il processo di calcolo degli attention scores rimane locale al layer in cui nasce.

Questa intuizione è implementata con la tecnica **query-key-value**, tre vettori rappresentativi che sono assegnati a ogni token e richiamati a seconda del ruolo che ricopre nel processo di calcolo degli attention-weights. Ogni token interagisce con gli altri per capire meglio se stesso. Gli è quindi assegnato un vettore **query** che presenta agli altri token per chiedergli se posseggono le informazioni che richiede. Gli altri token rispondono a questa domanda con il vettore **key** che, combinato con il query, comunica al mittente quale peso di attenzione assegnargli. Ne risulta una matrice di punteggi in cui sono contenuti gli attention score per ogni combinazione parola-parola. I risultati calcolati sono normalizzati dividendoli per la radice della dimensione delle query e delle key. Dopo l'applicazione di una softmax sono ottenuti gli attention weights.

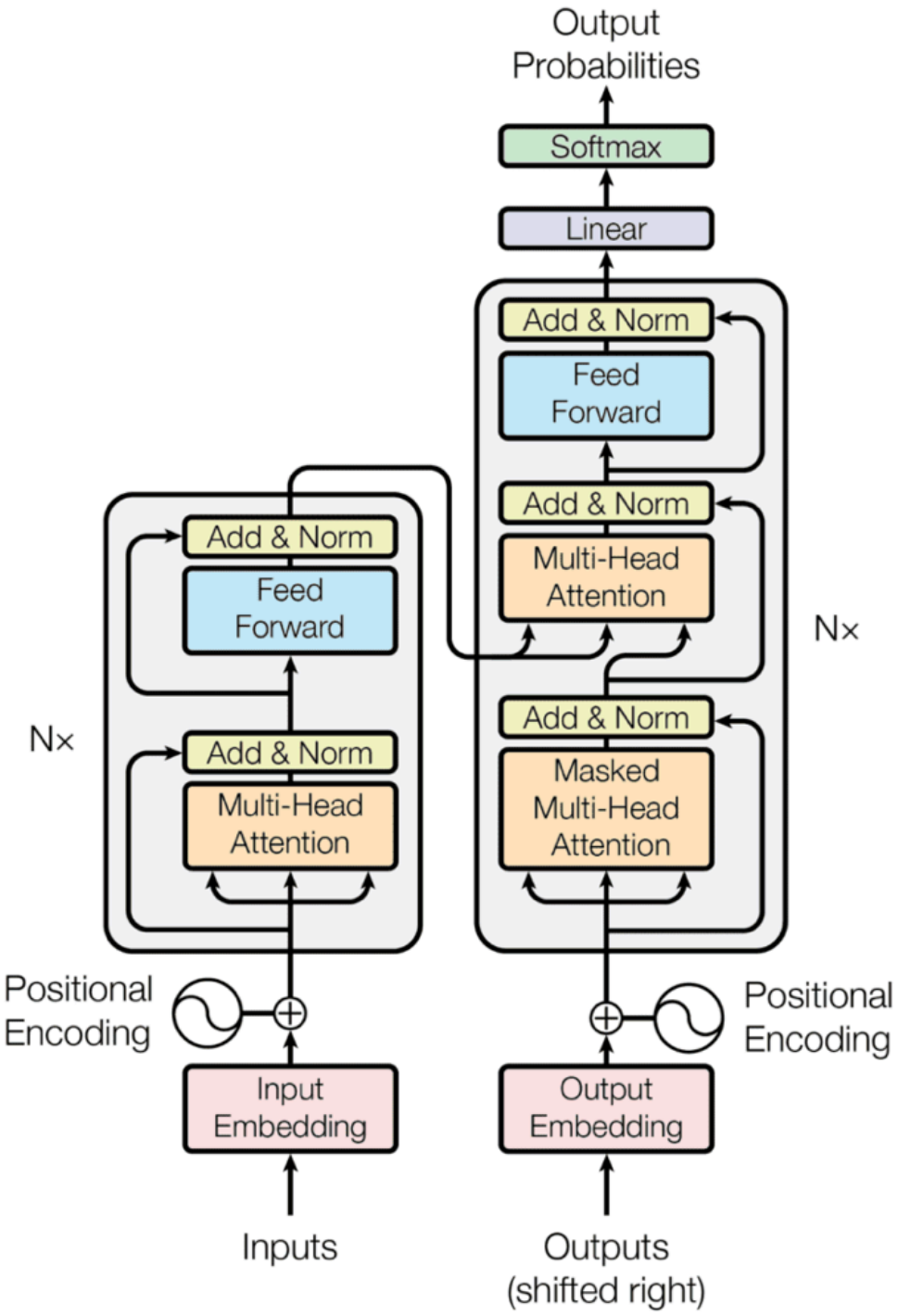
La somma dei **value** pesata sui punteggi calcolati risulta nell'attention output di ogni singolo token. Il value quindi è un valore che rappresenta la quantità di informazione che il token porta mentre la key specifica quale tipo di informazione possiede.



Masked Self-Attention

Nella fase di decodifica il meccanismo di self-attention funziona diversamente. I token sono generati uno alla volta quindi non sono disponibili immediatamente per interagire tra loro. Il decoder non conosce il futuro e quindi guarda solo ai token precedentemente generati per generare quello corrente. Un meccanismo che escluda i token futuri è necessario perchè durante il training si è a conoscenza dell'intera frase target e si vuole proibire al decoder di sbirciare. Per questo la **masked self-attention** maschera i token futuri.

Architettura

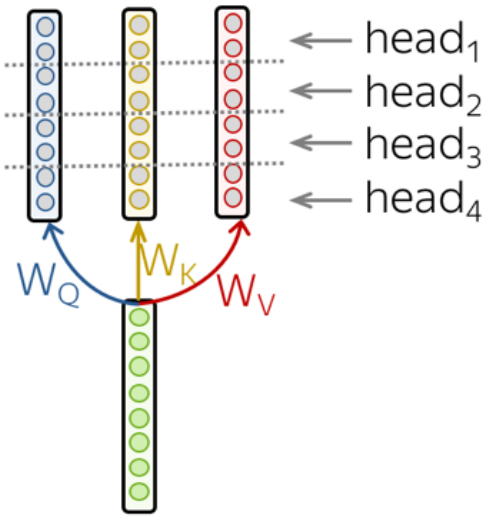


L'architettura dei transformer è formata da due blocchi principali: a sinistra il blocco di codifica, a destra quello di decodifica. Questi blocchi possono essere ripetuti quante volte si desidera.

Come già spiegato il blocco di codifica fa uso di un layer di self-attention il cui output è consegnato a un **feed forward network**, un blocco formato da due layer lineari separati da un'attivazione ReLU non lineare.

I layer **Add & Norm** applicano le connessioni residue all'output e lo normalizzano. Le connessioni residue ripropongono l'input di un layer aggiungendolo al suo output in modo da limitare il problema dei vanishing gradients.

Salta all'occhio come i layer di attenzione siano chiamati **Multi-Head**. Questo perchè più processi di attenzione, detti head, sono eseguiti in parallelo e i risultati sono poi combinati. I vettori di query, key e value sono divisi in più parti che sono singolarmente assegnate a una delle head del meccanismo. In questo modo non si verifica alcun aumento di dimensione nel modello. Ogni head impara qualcosa di diverso in modo da dare all'encoder più potenza di rappresentazione.



Entrambi i blocchi ricevono in input degli **embedding** combinati con una codifica posizionale. Gli embedding forniscono una rappresentazione vettoriale numerica delle parole mentre il **positional encoding** è fondamentale per comunicare all'architettura in che ordine sono posizionate.

Nel blocco di decoding, si trova anche un secondo layer di attenzione detto **encoder-decoder attention** che mette in atto un processo di multi-head attention guardando gli output dell'encoder oltre ai token di output precedentemente generati in modo da avere una migliore comprensione del contesto e offrire un risultato migliore. Le query e le key utilizzate sono quelle calcolate dall'encoder mentre i value sono quelli calcolati nello step di decoding precedente.

L'output finale è un insieme di distribuzioni di probabilità su un insieme di classi, ognuna rappreesentante una parola all'interno del vocabolario disponibile.

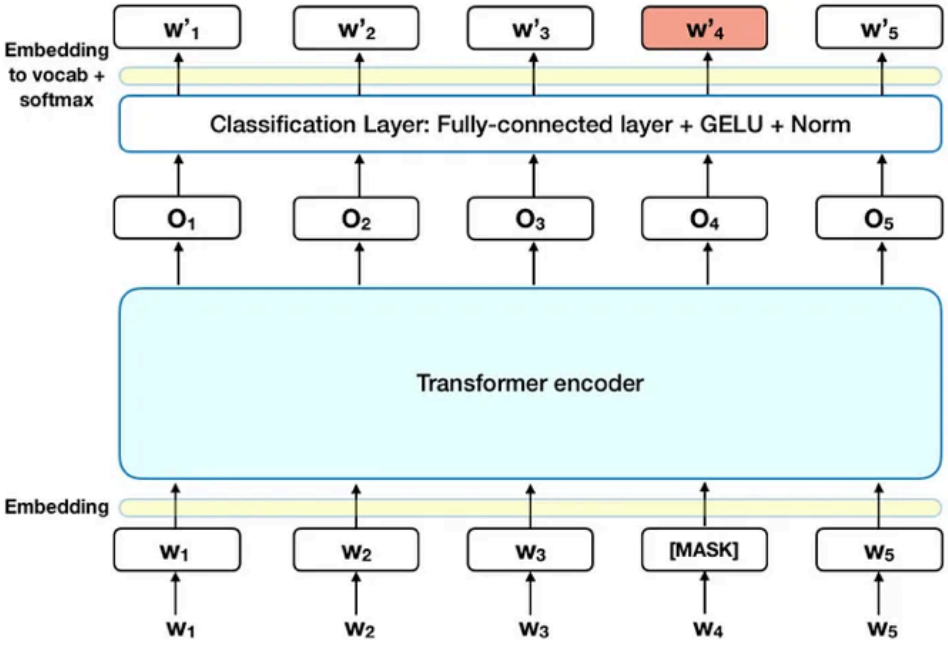
BERT

BERT (*Bi-Directional-Encoder Representations Transformers*) è un modello di linguaggio pre-addestrato che ha suscitato scalpore nella società del machine learning presentando risultati pioneristici su 11 diversi task NLP. Addestrato su **Wikipedia** (2.5 miliardi di parole) e il **BookCorpus** di Google (800 milioni di parole), BERT utilizza solo la componente di *encoding* del transformer per concentrarsi sulla comprensione del linguaggio naturale mentre il Transformer, che include anche una componente di *decoding*, dispone di ottime capacità nella generazione del linguaggio.

BERT impila molteplici componenti di encoding e utilizza un approccio bi-direzionale alla comprensione del linguaggio. Invece di analizzare il testo in modo sequenziale, per ogni parola viene preso in considerazione il **contesto**, ovvero ciò che si trova alla sua destra e alla sua sinistra.

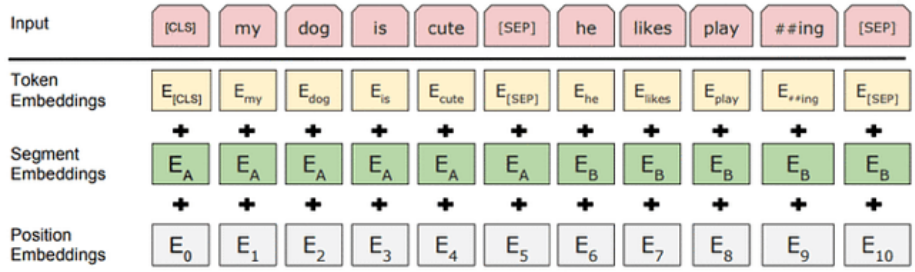
L'addestramento di BERT si divide in due fasi principali: * Nella fase di **pre-training** BERT è addestrato su una grande quantità di dati non annotati, con l'obiettivo di apprendere gli **embedding contestuali**, rappresentazioni delle parole che tengono conto del contesto in cui sono situate. * Nella fase di ***fine-tuning*** i parametri di BERT sono aggiornati per ottenere performance migliori nella risoluzione di task specifici come la text summarization (BERTSummarizer). Sono utilizzati dati annotati relativi al problema di interesse.

Uno dei punti chiave di questa architettura è una strategia chiamata **Masked Language Model (MLM)**. Durante la fase di pre-training, prima di dare delle frasi in pasto a BERT, il 15% delle parole sono rimpiazzate con un token **[MASK]**. Il modello prova a predire il valore originale delle parole mascherate osservando il contesto. Allo stack di encoding viene succeduto un *classification layer* che ha il compito di generare, con l'utilizzo di una softmax, le probabilità che i token mascherati combacino con una certa parola nel vocabolario.



BERT impara a comprendere la relazione tra frasi tramite la **Next Sentence Prediction (NSP)**. Il modello è addestrato su coppie di frasi che, nel 50% dei casi, si susseguono nel documento originale mentre l'altra metà è costituita da frasi sconnesse tra loro. Un token *[CLS]* viene inserito all'inizio della sequenza in input mentre dei token *[SEP]* sono posizionati alla fine delle due frasi. A ogni token è aggiunto l'embedding della frase di cui fa parte e un embedding posizionale per specificare la sua posizione nella sequenza

Per predire se la seconda frase è connessa alla prima, l'input viene processato dal modello e l'output del token *[CLS]* è trasformato in un vettore 2x1 con un layer di classificazione. Con una softmax viene calcolata la probabilità che le due frasi siano connesse.



BERT attua MLM e NSP contemporaneamente per apprendere le relazioni tra parole all'interno di una frase ma anche i collegamenti che ci sono tra diverse frasi.

Valutazione Pratica

Il presente rapporto illustra il testing condotto personalmente sulla piattaforma BERT al fine di effettuare extractive text summarization degli articoli contenuti nel dataset *Cornell Newsroom*. Dopo una fase di text processing sono state scelte 10 sintesi estrattive su cui testare e valutare il modello. Le predizioni sono state ottenute limitandole ad avere lo stesso rateo di compressione che hanno le sintesi target rispetto all'articolo originale in modo da non inquinare i risultati delle valutazioni.

I riassunti predetti dal modello sono stati paragonati con le sintesi target già presenti nel dataset per calcolare il loro punteggio **ROUGE**, un insieme di metriche utilizzate per valutare software dedicati all'automatic summarization. Sono state prese in considerazione:

- **ROUGE-1**: fa riferimento all'overlap di uni-grammi (singole parole) tra la predizione e il target
- **ROUGE-2**: si basa sull'overlap di bi-grammi (parole consecutive) tra predizione e target
- **ROUGE-L**: prende in considerazione la più lunga sotto-sequenza di parole in comune tra predizione e target

Metric	Summary 1	Summary 2	Summary 3	Summary 4	Summary 5	Summary 6	Summary 7	Summary 8	Summary 9	Summary 10
rouge-1 (F-Score)	0.424658	0.772277	0.275229	0.51145	0.153846	0.044444	0.396947	0.170732	0.181818	0.520548
rouge-2 (F-Score)	0.341463	0.740741	0.124197	0.40000	0.000000	0.000000	0.277778	0.020619	0.000000	0.432749
rouge-l (F-Score)	0.424658	0.772277	0.262997	0.48855	0.115385	0.044444	0.381679	0.146341	0.181818	0.520548

Metriche come ROUGE offrono misure quantitative ma spesso falliscono nel catturare la vera essenza di un riassunto ben fatto. ROUGE, dipendendo dalla presenza delle stesse esatte parole in entrambe le sintesi, manca di considerare la semantica dei loro contenuti. **BERTScore** sfrutta gli embedding contestuali del modello BERT per catturare le similarità semantiche che le metriche tradizionali basate su *n-grammi* falliscono nel catturare. Con un **F1-score** medio di 0,8664 è evidente come i risultati ottenuti siano migliori di come suggeriscono le misurazioni effettuate con ROUGE.

	Summary 1	**Summary 2**	**Summary 3**	**Summary 4**	**Summary 5**	**Summary 6**	**Summary 7**	**Summary 8**	**Summary 9**	**Summary 10**
F1-score	0.8632	0.9585	0.8277	0.8846	0.8256	0.8115	0.8756	0.8447	0.8719	0.9006

Per effettuare una valutazione finale è stato scelto il tool di *OpenAI* che fa uso di **GPT 3.5**, il famoso **Chat GPT**. Informato con le dovute istruzioni, il tool è stato utilizzato per valutare 4 caratteristiche dei riassunti generati da BERT:

- **Rilevanza:** Selezione delle informazioni salienti dalla sorgente. La sintesi deve contenere solo le informazioni più importanti del documento originale
- **Coerenza:** Il riassunto deve essere ben strutturato e organizzato. Non deve essere un insieme di informazioni ma deve costruire un discorso coerente a partire dalla prima frase fino all'ultima
- **Concordanza:** Un riassunto, per essere concordante con il documento originale, deve riportare solo fatti e affermazioni che sono contenute in esso
- **Scorrevolezza:** La qualità del riassunto in termini di grammatica, punteggiatura, scelta delle parole e struttura delle frasi

A tutte le metriche è stato assegnato un punteggio che varia da 1 a 5 tranne che per la scorrevolezza che ha ricevuto un punteggio da 1 a 3.

	Summary 1	**Summary 2**	**Summary 3**	**Summary 4**	**Summary 5**	**Summary 6**	**Summary 7**	**Summary 8**	**Summary 9**	**Summary 10**
Rilevanza	2	2	4	4	2	5	3	2	3	3
Coerenza	2	2	3	4	3	5	4	4	4	4
Concordanza	4	5	4	4	5	5	5	5	5	5
Scorrevolezza	2	2	3	3	3	3	3	3	3	3

	Rilevanza	**Coerenza**	**Concordanza**	**Scorrevolezza**
Punteggi Medi	3	3.5	4.7	2.8

Come emerge dai punteggi medi, i riassunti estrattivi ottengono ottimi risultati nella concordanza e nella scorrevolezza. Estraendo frasi dall'articolo originale è molto difficile introdurre errori fattuali o grammaticali. Per quanto riguarda rilevanza e scorrevolezza si è presentato qualche problema in più. Il modello non è sempre riuscito a includere tutte le informazioni salienti dell'articolo, a volte omettendo frasi essenziali per la comprensione dell'articolo.

Webgrafia

Sistemi Intelligenti

Web Scraping vs Web Crawling: https://www.baeldung.com/cs/web-crawling-vs-web-scraping

Scrapy – Settings https://www.geeksforgeeks.org/scrapy-settings/

Machine Learning

Understand Random Forest Algorithms With Examples (Updated 2023): https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Cos'è Random Forest?: https://www.ibm.com/it-it/topics/random-forest

Naive Bayes Classifier https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Natural Language Processing with Restaurant Reviews (Part 1): https://medium.com/analytics-vidhya/natural-language-processing-with-restaurant-reviews-part-1-46e228585a32

Natural Language Processing with Restaurant Reviews (Part 2): https://medium.com/analytics-vidhya/natural-language-processing-with-restaurant-reviews-part-2-ad240d1a7393

Natural Language Processing with Restaurant Reviews (Part 3): https://medium.com/analytics-vidhya/natural-language-processing-with-restaurant-reviews-part-3-2e08da61b8e5

A Beginner's Guide to Random Forest Hyperparameter Tuning: https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Deep Learning

Extractive Text Summarization with Deep Learning https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1273&context=cpesp

A Study on Neural Network Modeling Techniques for Automatic Document Summarization https://uu.diva-portal.org/smash/get/diva2:1365770/FULLTEXT01.pdf

Understanding Automatic Text Summarization-2: Abstractive Methods https://towardsdatascience.com/understanding-automatic-text-summarization-2-abstractive-methods-7099fa8656fe

A Survey of Text Summarization Extractive Techniques
https://www.researchgate.net/publication/228619779*A*Survey*of*Text*Summarization*Extractive_Techniques

Approaches to Text Summarization: An Overview https://www.kdnuggets.com/2019/01/approaches-text-summarization-overview.html

Comprehensive Guide to Text Summarization using Deep Learning in Python https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

Recurrent Neural Networks (RNNs) https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

Backpropagation Through Time (BPTT): Explained With Derivations https://www.pycodemates.com/2023/08/backpropagation-through-time-explained-with-derivations.html?utm_content=cmp-true **Understanding the attention mechanism in sequence models**
https://www.jeremyjordan.me/attention/

Sequence to Sequence (seq2seq) and Attention https://lena-voita.github.io/nlp*course/seq2seq*and_attention.html

Transformer: Attention is All You Need https://lena-voita.github.io/nlp*course/seq2seq*and_attention.html

Illustrated Guide to Transformers- Step by Step Explanation https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

How to Evaluate A Summarization Task https://github.com/openai/openai-cookbook/blob/main/examples/evaluation/How*to*eval*abstractive*summarization.ipynb

BERT Extractive Summarizer https://github.com/dmmiller612/bert-extractive-summarizer

BERT Explained: State of the art language model for NLP https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-

f8b21a9b6270