Università degli studi di Milano-Bicocca

Scuola di Economia e Statistica
Corso di Laurea in
STATISTICA E GESTIONE DELLE INFORMAZIONI

Tokenizzatori in Azione: Confronto e Applicazione per la Classificazione di Notizie in Diverse Metriche

Sara Borello
Matr. N. 882793

Keita Jacopo Viganò
Matr. N. 870980

ANNO ACCADEMICO 2022/2023

Indice

Introduzione

Nel campo del Natural Language Processing (NLP), la tokenizzazione riveste un ruolo cruciale, agendo come il primo passo essenziale nella trasformazione del linguaggio naturale in una forma che può essere elaborata computazionalmente. Questo processo implica la scomposizione del testo in unità più piccole, note come token, che possono essere parole, frasi o caratteri. La tokenizzazione non è un semplice atto di divisione del testo, ma un processo sofisticato che stabilisce le basi per ulteriori analisi linguistiche, permettendo ai computer di ”comprendere” e manipolare il linguaggio umano in un formato strutturato. La complessità della tokenizzazione varia significativamente a seconda della lingua e del contesto. In lingue come l'inglese, può sembrare relativamente semplice, basandosi su spazi bianchi e punteggiatura, ma in lingue senza separatori chiari tra le parole, come il cinese o il giapponese, il processo diventa notevolmente più complesso. Inoltre, la tokenizzazione deve gestire sfide come il trattamento di abbreviazioni, numeri, e diverse forme di testo specializzato. Dopo il processo di tokenizzazione, ogni token generato viene trasformato in rappresentazioni numeriche, un passaggio fondamentale per ulteriori elaborazioni in NLP. Tra le tecniche più rilevanti in questo contesto figurano i Word Embeddings, che giocano un ruolo cruciale nella cattura delle sfumature linguistiche. Questi embeddings trasformano i token in vettori densi in uno spazio multidimensionale, dove ogni dimensione rappresenta un aspetto del significato del token. Ciò consente di catturare non solo la frequenza delle parole, ma anche le loro relazioni semantiche e sintattiche nel contesto in cui appaiono. Ad esempio, parole con significati simili tendono ad avere embeddings che sono vicini nello spazio vettoriale, permettendo al modello di cogliere le somiglianze e le differenze nel significato delle parole. La tokenizzazione ad alto livello va oltre la semplice divisione del testo in parole o frasi. Si tratta di comprendere e segmentare il testo in unità più significative, che possono includere frasi idiomatiche, espressioni colloquiali, jargon tecnico, e altre costruzioni linguistiche complesse. Questo tipo di tokenizzazione è particolarmente importante perché il linguaggio umano è intrinsecamente ambiguo e ricco di sfumature. Ad esempio, la stessa parola può avere significati diversi a seconda del contesto, e la tokenizzazione ad alto livello aiuta a catturare queste sfumature. In applicazioni come l'analisi

del sentimento, la tokenizzazione ad alto livello permette di catturare espressioni che trasmettono emozioni o opinioni, che potrebbero essere perse con una tokenizzazione più basilare. Nella traduzione automatica, comprendere il contesto e le espressioni idiomatiche è essenziale per traduzioni accurate e naturali. Anche nella generazione di testo, come nella scrittura automatica di articoli o nella creazione di dialoghi per assistenti virtuali, una comprensione profonda delle strutture linguistiche attraverso una tokenizzazione avanzata consente di produrre testi più fluidi, naturali e coerenti.

1. Capitolo 1

1.1 Tokenizzazione

La tokenizzazione nel Processamento del Linguaggio Naturale è la procedura mediante la quale un testo viene suddiviso in unità più piccole, denominate ”token”. Un token può essere una parola, una frase, o anche un singolo carattere, a seconda dell'obiettivo specifico dell'attività di NLP. Questo processo è essenziale per convertire il testo, che di per sé è un formato non strutturato, in dati strutturati che possono essere processati dai modelli di NLP. I token sono quindi le unità atomiche nel NLP, che fungono da ponte tra il linguaggio naturale e la sua rappresentazione numerica che i modelli possono elaborare. Questa conversione è fondamentale perché i modelli computazionali possono elaborare solo dati numerici[1].

Il ”token” è definito rigorosamente come una sequenza contigua e non vuota di grafemi o fonemi all'interno di un documento. Questa definizione, fornita dal Morphological Annotation Framework (MAF) ISO standard, serve come un'approssimazione per le unità linguistiche, che sono spesso difficili da identificare o definire in modo consistente a causa di una vasta gamma di fenomeni linguistici come contrazioni, composti, derivati morfologici e varie classi di entità nominate.

1.1.1 Livelli di tokenizzazione

Nel trattamento automatico del linguaggio, la tokenizzazione gioca un ruolo cruciale nella determinazione della granularità con cui il testo viene suddiviso e analizzato. Oltre ai tradizionali livelli di tokenizzazione quali frasi, parole, sillabe e caratteri, è importante considerare la tokenizzazione a livello di unità sublessicali come morfemi e fonemi, specialmente per le lingue che condividono un alfabeto o in presenza di cognomi, parole in prestito e parole morfologicamente complesse. Questo aspetto diventa fondamentale in contesti come la traduzione automatica, dove

[1] Awaldeep Singh. ≪Hugging Face: Understanding tokenizers≫. In: *Medium* (2022).

la capacità di tradurre unità sublessicali consente una maggiore flessibilità e accuratezza, specialmente per parole rare o nuove. In tale contesto, la tokenizzazione a livello sublessicale permette al sistema di apprendere traduzioni trasparenti e generalizzare questa conoscenza per tradurre e produrre parole non viste[2].

Granularità	**Esempio di Input**	**Risultato della Tokenizzazione**
Frase	”Questa è una frase.”	[”Questa è una frase.”]
Parola	”Questa è una frase.”	[”Questa”, ”è”, ”una”, ”frase.”]
Sillaba	”Parola”	[”Pa”, ”ro”, ”la”]
Carattere	”Parola”	[”P”, ”a”, ”r”, ”o”, ”l”, ”a”]
Sublessicale	”Sonnensystem”	[”Sonne”, ”system”]

Tabella 1.1: Esempio di differenti granularità di Tokenizzazione

1.1.2 Problemi della tokenizzazione

Nella sua forma più semplice, la tokenizzazione si basa sulla segmentazione del testo in parole, utilizzando gli spazi bianchi come separatori. Questo metodo è relativamente efficace per lingue come l'inglese, dove la maggior parte delle parole è separata da spazi e la struttura morfologica è meno complessa. Lingue come l'arabo, il finlandese o il turco, caratterizzate da una ricca morfologia, richiedono approcci di tokenizzazione più sofisticati. In queste lingue, una singola parola può contenere un'abbondanza di informazioni grammaticali, come il genere, il numero, il caso, e la persona.

Per gestire questa complessità, si ricorre a metodi di tokenizzazione avanzati che possono scomporre le parole in morfemi, le più piccole unità di significato, permettendo ai modelli di apprendimento automatico di elaborare e comprendere meglio il testo. Inoltre, in specifici ambiti applicativi come la classificazione delle notizie o l'analisi del sentiment, si presentano ulteriori sfide. In questi contesti, il linguaggio può essere permeato da slang, gergo, o forme troncate, che

[2]CAGRI TORAMAN et al. ≪Impact of Tokenization on Language Models: An Analysis for Turkish≫. In: *Aselsan Research Center, Turkey* (2022).

una tokenizzazione elementare non è in grado di catturare efficacemente. La presenza di queste espressioni colloquiali, che spesso si discostano dalle norme grammaticali standard, richiede un approccio più flessibile e contestualizzato alla tokenizzazione, in grado di riconoscere e interpretare variazioni linguistiche non convenzionali.

Una delle maggiori sfide nella tokenizzazione, oltre alla complessità morfologica delle lingue, è la gestione della semantica. La semantica si riferisce al significato e all'interpretazione delle parole e delle frasi all'interno di un contesto specifico. La tokenizzazione semplice, che si basa principalmente sulla divisione del testo in parole o frasi basandosi su indicatori come gli spazi bianchi, spesso non riesce a catturare la complessità semantica del linguaggio. Per esempio, la stessa parola può avere significati differenti in contesti diversi, e la tokenizzazione semplice non è in grado di discernere queste sfumature.

Un "token di alto livello" in elaborazione del linguaggio naturale (NLP) si riferisce a un'unità di testo che va oltre la semplice suddivisione basata su parole o caratteri. A differenza dei token standard, che si limitano a segmenti di parole o singoli caratteri, spesso perdendo le sfumature di significato, i token di alto livello mantengono l'integrità semantica del corpus.

1.2 Tokenizer

La scelta della metodologia di tokenizzazione è cruciale, in quanto influisce direttamente sul modo in cui i modelli di NLP interpretano e analizzano il testo. Ci sono diverse metodologie di tokenizzazione, ognuna con le sue caratteristiche e applicazioni specifiche[3].

1.2.1 Metodi basati su linguistic concepts

L'approccio basato sulla conoscenza linguistica e sui concetti tradizionali si riferisce a metodi di tokenizzazione che richiedono una comprensione esplicita delle strutture della lingua e utilizzano questo sapere per segmentare il testo. Questi metodi sono spesso radicati nella linguistica computazionale e possono essere molto sofisticati, richiedendo una progettazione dettagliata e

[3]Sabrina J. Mielke et al. ≪Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP≫. in: *BigScience Workshop Tokenization Working Group* (2021).

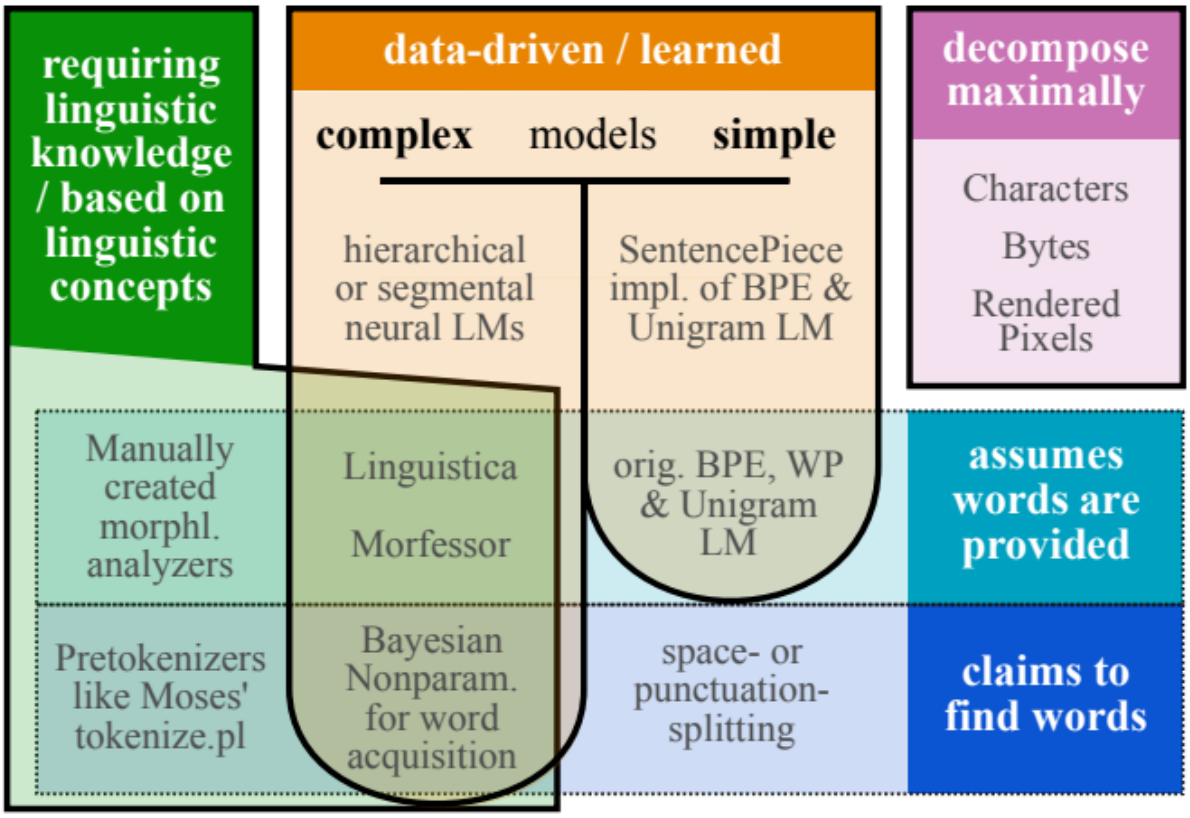

Figura 1.1: Una tassonomia degli algoritmi di segmentazione e tokenizzazione

spesso un'attenta manutenzione da parte di esperti linguisti. Gli analizzatori morfologici manuali sono un esempio di questi sistemi. Lavorano sulla base di un insieme di regole codificate manualmente che descrivono la morfologia di una lingua - cioè, come le parole si formano e si modificano per esprimere diversi significati grammaticali. Queste regole possono coprire la flessione delle parole per numero, caso, tempo, genere, ecc., nonché la derivazione, che riguarda la formazione di nuove parole attraverso prefissi, suffissi e altri mezzi. Ad esempio, un analizzatore morfologico per l'inglese potrebbe riconoscere che "books" è il plurale di "book", o che "running" è una forma del verbo "run". Questi sistemi tendono ad essere altamente accurati per le lingue per le quali sono stati progettati, ma richiedono un notevole investimento iniziale in termini di tempo e risorse, dato che le regole devono essere scritte a mano da esperti. Inoltre, tali analizzatori possono lottare nel gestire le eccezioni linguistiche e possono avere difficoltà ad adattarsi a nuovi vocaboli o usi della

lingua che non erano previsti al momento della loro creazione.

Per facilitare il lavoro degli analizzatori morfologici, si possono utilizzare i pretokenizzatori. Questi strumenti eseguono un primo passaggio di elaborazione del testo, applicando regole più semplici basate su spazi e punteggiatura per dividere il testo in unità più gestibili. Ad esempio, il tokenizzatore di Moses, che è ampiamente utilizzato nel campo della traduzione automatica, applica un insieme di regole euristiche per separare le parole e i segni di punteggiatura in modo da preparare il testo per l'analisi successiva. Questo processo rende il testo più uniforme e può migliorare la performance degli analizzatori morfologici, riducendo la varietà e la complessità delle entrate che devono trattare.

I metodi di tokenizzazione data-driven possono essere distinti in base alla loro complessità e alla necessità di un vocabolario predefinito.

1.2.2 Metodi Data-Driven Complessi

I metodi di tokenizzazione data-driven complessi spesso non richiedono un vocabolario predefinito. Sono in grado di costruire e adattare il loro vocabolario durante il processo di apprendimento dal corpus di testo su cui vengono addestrati. Questi metodi includono reti neurali gerarchiche o segmentali che apprendono rappresentazioni di dati in maniera stratificata. Ogni strato in una rete gerarchica costruisce rappresentazioni sempre più astratte, mentre le reti segmentali sono addestrate a riconoscere e segmentare unità linguistiche all'interno di un flusso di testo continuo.

All'interno di questa categoria, ci sono strumenti come Morfessor, che operano in modo non supervisionato per identificare morfemi o altre unità sub-lexicali senza dipendere da un vocabolario fisso. Questi strumenti utilizzano modelli probabilistici per dedurre i confini dei token basandosi sulle proprietà statistiche dei dati di testo. Inoltre, metodi Bayesiani non parametrici si adattano e crescono con i dati, permettendo al vocabolario di espandersi naturalmente senza la necessità di una lista preconfezionata di parole o token.

1.2.3 Metodi Data-Driven Semplici

I metodi data-driven semplici, d'altra parte, spesso si basano su un vocabolario predefinito o lo costruiscono in una fase iniziale del loro processo. Questi includono tecniche come il Byte Pair Encoding (BPE) e modelli linguistici Unigram. BPE inizia con un vasto vocabolario di caratteri o byte e successivamente fonde le coppie più frequenti per formare nuovi token, mentre i modelli Unigram selezionano la segmentazione che massimizza la probabilità del corpus di testo dato un certo vocabolario di token. SentencePiece è un esempio di un modello semplice che può funzionare senza un vocabolario pre-tokenizzato; tuttavia, costruisce un vocabolario durante la sua inizializzazione. Questi metodi sono generalmente più veloci e meno computazionalmente intensivi rispetto ai metodi complessi e sono quindi utilizzati in ambienti dove la velocità e l'efficienza sono cruciali.

1.2.4 Metodi di decomposizione massima

Nell'ambito dell'elaborazione del linguaggio naturale, esistono approcci di tokenizzazione che si concentrano sulla decomposizione massima del testo, scomponendolo nelle sue unità costitutive più elementari. Questi metodi si distaccano dalla segmentazione basata su unità linguistiche come parole o morfemi, preferendo invece una destrutturazione del testo a un livello più fondamentale. Uno di questi approcci è la tokenizzazione a livello di caratteri, che divide il testo in singoli caratteri alfabeticamente o simbolicamente definiti. Questa metodologia si rivela particolarmente utile per lingue come il cinese, che non hanno separatori evidenti tra le parole, e per lingue con sistemi di scrittura complessi, dove la morfologia può essere altamente intricata. È anche impiegata in contesti di deep learning per consentire ai modelli di apprendere le relazioni tra i caratteri all'interno delle parole. Un altro metodo è la tokenizzazione a livello di byte. Questa tecnica va oltre la semplice suddivisione basata sui caratteri, considerando la rappresentazione binaria del testo. Indipendente dalla lingua, non richiede alcuna conoscenza preesistente delle regole grammaticali o morfologiche specifiche di una lingua. Questo la rende ideale per gestire testo in una varietà di lingue o in contesti in cui si debbano elaborare dati binari che non sono necessariamente testo. Infine, la tokenizzazione di pixel renderizzati, meno comune ma ugualmente

importante, implica convertire il testo in immagini e poi segmentare queste immagini in base ai pixel. Questo approccio è particolarmente rilevante nell'ambito della lettura ottica dei caratteri (OCR), dove si converte il testo stampato in dati digitali. I pixel che costituiscono i caratteri diventano l'unità fondamentale di tokenizzazione in questo processo. Questi metodi poiché operano a un livello di granularità molto fine, possono richiedere un numero maggiore di parametri per catturare le stesse informazioni che potrebbero essere rappresentate da token più grandi e più ricchi di significato, come le parole. Ciò comporta una maggiore esigenza di risorse di memoria e potenza di calcolo durante l'addestramento e l'inferenza, rendendo questi approcci più impegnativi per lo sviluppo di modelli di machine learning su vasta scala.

2. Capitolo 2

2.1 BPE

Il Byte Pair Encoding, abbreviato in BPE, si inserisce nel panorama delle tecniche di compressione dati come un metodo rivoluzionario, nato nell'ambito degli schemi macro da Storer e Szymanski nel 1982 e successivamente definito in maniera più formale da Philip Gage nel 1994. Sebbene concetti analoghi fossero stati esplorati in precedenza per lo studio delle lingue naturali e la complessità delle sequenze genetiche, il contributo di Gage ha segnato un punto di svolta nella storia di questa tecnica[4].

2.1.1 Funzionamento

Il BPE è un metodo di tokenizzazione originariamente sviluppato per la compressione di dati, ma che ha trovato un'applicazione estremamente efficace nella tokenizzazione per i modelli di linguaggio naturale. Il processo di BPE è caratterizzato da un approccio iterativo e rigoroso che inizia con una fase di pre-tokenizzazione e si estende fino all'addestramento su un corpus e alla creazione di un vocabolario ottimizzato.
Nella fase iniziale, BPE richiede una pre-tokenizzazione del testo.[5] Questo processo generalmente coinvolge la suddivisione del testo in unità di base come parole, spesso utilizzando separatori semplici come spazi e punteggiatura. Questa suddivisione iniziale è cruciale poiché stabilisce le unità di base su cui il BPE opererà. Successivamente, viene avviato il processo di addestramento di BPE sul un corpus di testo tokenizzato, durante il quale viene inizializzato un vocabolario di caratteri di base, che include ogni carattere unico presente nel corpus. Poi, in modo iterativo, BPE esamina il corpus per identificare le coppie di caratteri adiacenti che compaiono più frequentemente. Ad ogni iterazione, la coppia di caratteri più comune viene fusa insieme per formare un

[4] Vilém Zouhar et al. ≪A Formal Perspective on Byte-Pair Encoding≫. In: *ETH Zürich* (2023).

[5] Hugging Face. *Byte-Pair Encoding tokenization*. URL: `https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt`.

nuovo simbolo, che viene quindi aggiunto al vocabolario. Ad esempio, se ”e” e ”s” si verificano spesso uno accanto all'altro, ”es” diventa un nuovo token nel vocabolario. Questo processo di fusione viene ripetuto molte volte, ogni volta aggiungendo al vocabolario i nuovi simboli che rappresentano combinazioni di caratteri sempre più lunghe come nell'esempio riportato nella figura 2.1. Il processo continua fino al raggiungimento di un criterio di terminazione predefinito, che può essere un numero specifico di iterazioni o una dimensione massima del vocabolario. Il risultato è un vocabolario che comprende non solo caratteri singoli, ma anche una serie di sotto-parole o sequenze di caratteri che si verificano comunemente insieme. Questi nuovi simboli aiutano a ridurre la dimensione del vocabolario necessario per rappresentare efficacemente l'intero corpus di testo, mantenendo al contempo la capacità di coprire una vasta gamma di parole, incluse quelle che non sono state viste durante l'addestramento. Quando BPE è applicato a un nuovo testo, il testo viene prima pre-tokenizzato nello stesso modo del corpus di addestramento. Poi, utilizzando il vocabolario creato durante l'addestramento, il testo è suddiviso in token. Se una parola non è presente nel vocabolario, viene scomposta nelle sotto-unità più grandi possibili secondo il vocabolario BPE. Questo permette ai modelli di linguaggio di gestire efficacemente parole rare o sconosciute, suddividendole in pezzi familiari.

Iterazione	Token
1	a, a, a, b, d, a, a, a, b, a, c
2	aa, a, b, d, aa, a, b, a, c
3	aaa, b, d, aaa, b, a, c
4	aaa, b, d, aaa, b, ac

Tabella 2.1: Schema iterazioni nella Tokenizzazione BPE

2.1.2 Vantaggi e Svantaggi

Il Byte Pair Encoding (BPE) è una tecnica di tokenizzazione influente nel campo dell'elaborazione del linguaggio naturale (NLP), particolarmente per i modelli di machine learning. Offre

diversi vantaggi, ma presenta anche alcune limitazioni, specialmente quando si tratta di gestire lingue diverse.

Vantaggi di BPE

- Riduzione della Complessità del Vocabolario: BPE effettivamente riduce la dimensione del vocabolario richiesto per un modello di linguaggio, rendendo più gestibile l'elaborazione del testo. Questo è particolarmente utile per modelli di machine learning che possono essere gravati da set di dati molto grandi.

- Gestione delle Parole Rare o Sconosciute: Uno dei principali vantaggi di BPE è la sua capacità di trattare efficacemente parole rare o sconosciute. Decomponendo le parole in sottounità comuni, BPE consente ai modelli di linguaggio di interpretare e gestire parole che non sono state incontrate durante l'addestramento.

- Efficienza in Contesti Multilingue: BPE è particolarmente utile in applicazioni multilingue, come la traduzione automatica e il riconoscimento vocale. La sua flessibilità nel trattare variazioni linguistiche, inclusi caratteri non standard, lo rende un ottimo strumento per applicazioni che necessitano di trattare diverse lingue e dialetti.

Limitazioni di BPE

- Mancanza di Contesto e Semantica: BPE si concentra sulle frequenze delle coppie di simboli senza tenere conto del contesto o del significato delle parole. Questo può portare a rappresentazioni non ottimali per parole con significati complessi o in contesti dove la forma della parola cambia il suo significato.

- Sfide nelle Lingue con Morfologia Complessa: In lingue con strutture morfologiche ricche, come l'arabo o il finlandese, BPE potrebbe non catturare adeguatamente le variazioni morfologiche. Questo può incidere sulla qualità della tokenizzazione e, di conseguenza, sull'efficacia del modello di NLP.

- Equilibrio nella Dimensione del Vocabolario: La scelta della dimensione del vocabolario e del numero di operazioni di fusione in BPE è cruciale. Un vocabolario troppo grande

può portare a una frammentazione eccessiva, mentre uno troppo piccolo può risultare in una generalizzazione eccessiva. Questo equilibrio è particolarmente importante in contesti multilingue dove diverse lingue possono richiedere approcci diversi.

2.1.3 Utilizzo

Il Byte Pair Encoding (BPE) è attualmente utilizzato in diverse applicazioni nell'ambito dell'elaborazione del linguaggio naturale (NLP), grazie alla sua efficacia nel ridurre la complessità del vocabolario e nel gestire le parole rare o sconosciute. Queste caratteristiche lo rendono particolarmente utile per i modelli di linguaggio pre-addestrati, come GPT-3 e BERT. In questi modelli, BPE permette una rappresentazione compatta e versatile del testo, contribuendo significativamente alla loro capacità di elaborare e generare testo efficacemente. Una delle principali aree di applicazione di BPE è la traduzione automatica, dove la sua capacità di trattare variazioni linguistiche e caratteri non standard, come emoji o simboli, è particolarmente vantaggiosa. Questo lo rende una scelta eccellente per contesti multilingue. Inoltre, BPE trova impiego anche in sistemi di riconoscimento vocale, dove può migliorare l'accuratezza nella trascrizione di lingue e dialetti diversi. La tecnica BPE è anche utile nell'analisi di sentimenti e nella generazione di testo, dove può aiutare a catturare le sfumature linguistiche complesse. Ad esempio, in tokenizzazioni a livello di parola, spesso sono necessari token separati per diverse forme della stessa parola, come "eat/eating/eats". Con la tokenizzazione a livello di sotto-parola di BPE, è possibile catturare la radice e il suffisso separatamente, consentendo una maggiore flessibilità e precisione nell'analisi del testo[6].

2.2 Morfessor 2.0

Nel campo dell'elaborazione del linguaggio naturale, il ruolo della segmentazione morfologica è fondamentale per comprendere e processare efficacemente i testi. Morfessor emerge come una famiglia di metodi di apprendimento automatico probabilistico, sviluppati specificamente per questa finalità. Questo modello, attraverso una sofisticata analisi dei dati testuali non

[6]Mielke et al., ≪Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP≫.

annotati, identifica la segmentazione morfologica, ovvero la suddivisione delle parole in morfemi, le più piccole unità portatrici di significato in una lingua. La versione 2.0 di Morfessor rappresenta una significativa evoluzione rispetto al suo predecessore, Morfessor 1.0. Questa nuova versione non solo conserva le efficaci funzionalità del modello originale ma introduce anche miglioramenti notevoli, come l'apprendimento semi-supervisionato, l'online training e un integrated evaluation code.

L'obiettivo principale di Morfessor è quello di segmentare le parole in morfemi in modo non supervisionato. La prima versione di Morfessor, nota come Morfessor Baseline, è stata sviluppata da Creutz e Lagus nel 2002, seguita dalla sua implementazione software, Morfessor 1.0, nel 2005[7]. Successivamente, sono state sviluppate diverse varianti di Morfessor, ognuna mirata a migliorare alcuni aspetti dell'algoritmo originale, pur mantenendo la popolarità della versione Baseline come analizzatore morfologico generale.

Negli ultimi anni, Morfessor ha trovato applicazione in un'ampia gamma di lingue e contesti, dimostrando la sua efficacia in ambiti quali il riconoscimento vocale continuo a vasto vocabolario, la traduzione automatica e il recupero di informazioni vocali.

2.2.1 Composizione di Morfessor

Morfessor è caratterizzato principalmente da tre componenti: il modello in sé, la funzione di costo e gli algoritmi di addestramento e decodifica[8].

- Il **modello** Morfessor si compone di due parti principali: un lexicon e una grammatica. Il lexicon memorizza le proprietà delle costruzioni (unità di segmentazione più piccole del testo), mentre la grammatica determina come queste costruzioni possono essere combinate per formare i composti (unità più grandi, come parole o frasi). La grammatica di Morfessor si basa su due ipotesi fondamentali: la prima è che un composto è costituito da una o più costruzioni,

[7]Sami Virpioja, Peter Smit e Miko Kurimo. ≪Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline≫. In: *Department of Information and Computer Science, Aalto University* (2013).

[8]Peter Smit. ≪Morfessor 2.0: Toolkit for statistical morphological segmentation≫. In: *Department of Information and Computer Science, Aalto University* (2014).

e il limite massimo del numero di costruzioni in un composto è dato dal numero di atomi (le unità più piccole, come le lettere). La seconda ipotesi è che le costruzioni di un composto si verifichino in modo indipendente, un'assunzione che, nonostante sia probabilmente inesatta, consente l'utilizzo di algoritmi di addestramento molto efficienti.

- La **cost function** in Morfessor deriva dalla stima massima a posteriori (MAP) per il modello. Si compone di due parti: la verosimiglianza del modello e il prior. La verosimiglianza è derivata direttamente dalle ipotesi del modello menzionate sopra. Il prior, che determina la probabilità del lessico del modello, si ispira al principio della Minima Lunghezza di Descrizione (MDL), basandosi su schemi di compressione efficienti.

- L'algoritmo di **training** è caratterizzato da una ricerca "greedy" e locale. Inizia con un lessico iniziale, solitamente composto da tutte le forme di composti osservate nei dati di addestramento. Successivamente, seleziona un composto alla volta e cerca di trovare contemporaneamente la segmentazione ottimale per quel composto e il lessico ottimale, data la nuova segmentazione e le segmentazioni di tutti gli altri composti noti. L'addestramento avviene normalmente in batch, ma le versioni più recenti supportano anche l'addestramento online.

- Per la **decodifica**, ovvero la ricerca delle segmentazioni ottimali per nuove forme di composti senza modificare i parametri del modello, Morfessor applica una variazione dell'algoritmo di Viterbi.

2.2.2 Workflow

Il Workflow[9] di Morfessor inizia con la fase di inizializzazione, dove il modello viene configurato con un lexicon iniziale. Questo lessico, che include tutte le forme composte presenti nel dataset di addestramento, è la base da cui il modello inizia a costruire la sua comprensione delle strutture linguistiche. L'addestramento di Morfessor 2.0 segue un approccio iterativo. Durante questo processo, il modello elabora ogni forma composta nel dataset di addestramento, testando

[9] Virpioja, Smit e Kurimo, ≪Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline≫.

diverse segmentazioni possibili. La scelta della segmentazione ottimale per ciascuna forma composta si basa sulla minimizzazione del costo (Figura 2.1), un criterio che prende in considerazione sia la complessità del lexicon che la coerenza delle segmentazioni rispetto ai dati di addestramento. È qui che si manifesta la sottile interazione tra lessico e grammatica: mentre il lessico si evolve verso una forma più efficiente e compatta, la grammatica implicita nel modello guida la segmentazione in maniera coerente con i pattern linguistici osservati.

Uno degli aspetti più innovativi di Morfessor 2.0 è l'utilizzo dell'algoritmo di Viterbi esteso nella fase di decodifica. Dopo che il modello è stato addestrato e ottimizzato, esso è pronto per essere applicato a nuove forme composte. L'algoritmo di Viterbi esteso facilita l'identificazione delle segmentazioni ottimali per queste nuove forme, basandosi sulle conoscenze e sui pattern acquisiti durante l'addestramento. In questa fase, la grammatica, sebbene non esplicita, gioca un ruolo cruciale, influenzando la segmentazione in modo che sia allineata con le strutture linguistiche precedentemente osservate.

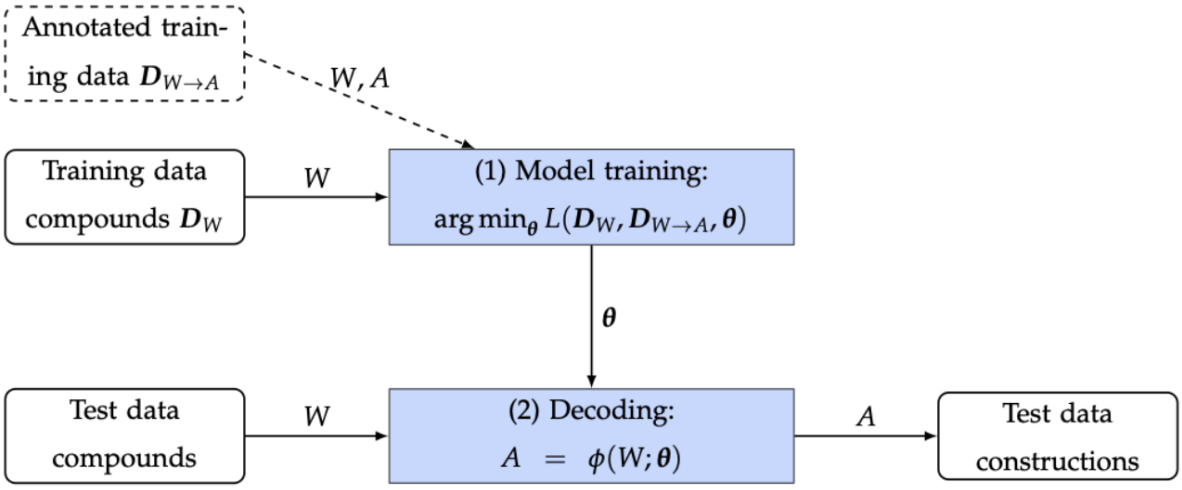

Figura 2.1: Workflow di Morfessor 2.0

2.2.3 Vantaggi e Svantaggi

Vantaggi di Morfessor

- Gestione delle Lingue Morfologicamente Ricche: Morfessor eccelle nell'analizzare lingue con una morfologia complessa, come il finlandese o il turco. È in grado di scomporre efficacemente le parole in morfemi, rendendolo ideale per applicazioni in queste lingue.

- Apprendimento Automatico: Come algoritmo basato su machine learning, Morfessor può apprendere direttamente dai dati, senza la necessità di regole morfologiche predefinite, rendendolo flessibile e adattabile.

- Riduzione della Dimensione del Vocabolario: Aiuta a ridurre la dimensione del vocabolario mantenendo una rappresentazione ricca delle parole, utile per modelli di linguaggio e altre applicazioni di NLP.

- Supporto Semi-Supervisionato: Morfessor supporta l'apprendimento semi-supervisionato, consentendo l'utilizzo di un mix di dati annotati e non annotati, che può migliorare la qualità della segmentazione.

Limitazioni di Morfessor

- Complessità per Lingue con Morfologia Semplice: Per lingue con una struttura morfologica meno complessa, come l'inglese, Morfessor può essere meno efficace o necessario rispetto ad altri metodi di tokenizzazione più semplici.

- Dipendenza dal Corpus di Addestramento: La qualità delle segmentazioni di Morfessor dipende fortemente dalla rappresentatività e dalla dimensione del corpus di addestramento. Un corpus limitato o di bassa qualità può portare a risultati subottimali.

- Complessità Computazionale: Il processo di addestramento e segmentazione di Morfessor può essere computazionalmente più intenso rispetto ad altri metodi di tokenizzazione, specialmente per grandi volumi di dati.

- Necessità di Regolazione dei Parametri: Morfessor richiede un'accurata regolazione dei parametri per bilanciare tra sovra e sotto-segmentazione, il che può richiedere tempo e sperimentazione per ottimizzare.

2.3 Moses tokenizer

Moses è un sistema di traduzione automatica statistica (Statistical Machine Translation, SMT) open-source ampiamente utilizzato[10]. È un'implementazione dell'approccio statistico alla traduzione automatica, che è attualmente l'approccio dominante nel campo. Moses viene addestrato su grandi quantità di dati paralleli (testo in due lingue diverse, allineato a livello di frase) e su ancora maggiori quantità di dati monolingui, da cui il sistema impara come dovrebbe apparire la lingua target. Il sistema Moses è composto principalmente da due componenti:

- **Training**: il testo viene tokenizzato e i token vengono convertiti in un formato standard. Si applicano euristiche per rimuovere coppie di frasi mal allineate e si escludono frasi eccessivamente lunghe. Le frasi parallele vengono poi allineate a livello di parola, solitamente con GIZA++, un insieme di modelli statistici sviluppati da IBM negli anni '80. Questi allineamenti sono fondamentali per estrarre traduzioni frase-frase o regole gerarchiche e per stimare le probabilità su scala corpus. Un elemento cruciale è il modello linguistico, costruito con dati monolingui nella lingua di destinazione, che aiuta il decodificatore a garantire la fluidità dell'output. Per la costruzione di questo modello, Moses si affida a strumenti esterni. L'ultimo passo è il tuning, dove i diversi modelli statistici vengono ponderati per ottenere le migliori traduzioni possibili. Moses include implementazioni degli algoritmi di tuning più popolari.

- **Decoding**: ha il compito di trovare la frase con il punteggio più alto nella lingua target, corrispondente a una data frase sorgente, secondo il modello di traduzione. È possibile

[10]Philipp Koehn et al. ≪Moses: Open Source Toolkit for Statistical Machine Translation≫. In: *Association for Computa- tional Linguistics* (2007).

che il decoder fornisca anche una lista ordinata di candidati per la traduzione e vari tipi di informazioni su come ha preso la sua decisione, come le corrispondenze frase-frase utilizzat

Nel contesto di Moses, la tokenizzazione è un passo cruciale nel processo di addestramento del modello di traduzione. Il Moses Tokenizer è responsabile della suddivisione del testo in token, che sono le unità più piccole di testo che hanno significato semantico o sintattico. Questo processo è fondamentale per preparare i dati per le fasi successive di allineamento delle parole e estrazione delle frasi.

2.3.1 Processo di Tokenizzazione

Il Moses tokenizer[11], essendo parte del più ampio toolkit Moses per la traduzione automatica, non è inizialmente concepito per un uso diretto in ambienti di programmazione come Python. Tuttavia, per superare questa limitazione e rendere il Moses tokenizer accessibile agli sviluppatori Python, si ricorre all'uso di un "wrapper" Python. Un wrapper è una sorta di strato intermedio o interfaccia software che permette l'integrazione e l'interazione di un programma (in questo caso, il Moses tokenizer) in un ambiente diverso da quello per cui è stato originariamente progettato. In pratica, il wrapper Python "avvolge" il Moses tokenizer, consentendo agli sviluppatori di utilizzarlo all'interno di applicazioni Python senza dover interagire direttamente con il codice Perl sottostante. Questo approccio apre la strada a una più ampia flessibilità nell'uso del Moses tokenizer. Per esempio, all'interno di un'applicazione Python, è possibile richiamare il tokenizer, specificare la lingua del testo sorgente e applicare le regole di tokenizzazione pertinenti [12]. Queste regole (alcune mostrate nella Figura 2.2), implementate nello script Perl tokenizer.perl, sono basate su spazi, punteggiatura e convenzioni linguistiche specifiche per ciascuna lingua, rendendo il processo di tokenizzazione sia accurato che versatile. Durante la tokenizzazione, una delle funzioni principali dello script è quella di gestire la punteggiatura. Ad esempio, può aggiungere o rimuovere spazi attorno ai segni di punteggiatura per standardizzare il formato del testo. Questo è particolarmente importante per la traduzione automatica, dove la coerenza e la precisione del

[11] Philipp Koehn. ≪Moses: Statistical Machine Translation System, User Manual and Code Guide≫. In: *University of Edinburgh* (2022).

[12] Moses on Github:https://github.com/mosessmt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl

formato del testo possono influire significativamente sulla qualità della traduzione. Alla fine del processo di tokenizzazione, il testo viene trasformato in una serie di token, con ogni token separato da spazi. Questa struttura standardizzata facilita enormemente l'analisi successiva e l'elaborazione del testo da parte di sistemi di traduzione automatica o di altre applicazioni di elaborazione del linguaggio naturale.

Figura 2.2: Regole di Moses

2.3.2 Vantaggi e Svantaggi

Il Moses Tokenizer offre un insieme di vantaggi e svantaggi che lo rendono un componente significativo nell'ambito della tokenizzazione nell'elaborazione del linguaggio naturale e della traduzione automatica. Uno dei suoi principali punti di forza risiede nella sua capacità di personalizzare le regole di tokenizzazione per adattarsi alle specifiche lingue, consentendo una tokenizzazione accurata e coerente. Questa flessibilità lo rende adatto a una vasta gamma di lingue e contesti linguistici. Un altro vantaggio rilevante è la sua capacità di standardizzare il formato del testo tokenizzato. Ogni token viene separato da spazi, semplificando notevolmente l'elaborazione dei dati tokenizzati e l'ulteriore analisi linguistica. Inoltre, il Moses Tokenizer è in grado di gestire la punteggiatura, incluso l'aggiungere o rimuovere spazi attorno ai segni di punteggiatura per garantire una tokenizzazione coerente. Tuttavia, il Moses Tokenizer presenta anche alcune limitazioni. La sua configurazione e utilizzo richiedono familiarità con il linguaggio di program-

mazione Perl, in cui è implementato. Questo potrebbe costituire un ostacolo per gli sviluppatori non esperti in Perl. Inoltre, mentre il tokenizzatore è altamente personalizzabile per diverse lingue, potrebbe non essere altrettanto adattabile a linguaggi con regole di tokenizzazione molto diverse o complesse.

Infine, è importante notare che il Moses Tokenizer è stato originariamente progettato per l'addestramento di modelli di traduzione automatica e potrebbe non essere la scelta ottimale per applicazioni che richiedono prestazioni in tempo reale o l'integrazione in scenari con requisiti di prestazioni molto elevati.

2.4 Unigram LM

Il tokenizzatore UnigramLM, introdotto da Taku Kudo[13] come parte del progetto SentencePiece di Google, ha segnato un passo significativo nello sviluppo di strumenti per l'elaborazione NLP. Questo approccio innovativo è apprezzato per la sua flessibilità e adattabilità a un'ampia varietà di lingue, inclusi quelli con vocabolari ampi e in continua evoluzione. Una caratteristica notevole del tokenizzatore Unigram è il suo metodo basato su probabilità, che permette di gestire efficacemente sia lingue ampiamente diffuse sia quelle meno comuni, inclusi i linguaggi con strutture morfologiche complesse o inusuali. Tale versatilità lo rende un'opzione ideale in diverse applicazioni di NLP, da quelle che richiedono traduzione automatica accurata a quelle orientate alla generazione di testo e alla comprensione del linguaggio naturale. Il contributo di Kudo con il tokenizzatore Unigram ha aperto nuove possibilità nel campo dell'NLP, permettendo lo sviluppo di modelli di machine learning più robusti e affidabili, specialmente per lingue meno documentate o linguisticamente complesse.

2.4.1 Funzionamento

La tokenizzazione Unigram si basa su un modello di linguaggio chiamato "Unigram Language Model" (modello di linguaggio unigramma). In questo modello, ciascun token o sotto-unità

[13]Taku Kudo. ≪Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates≫. In: *Google, Inc.* (2018).

lessicale è considerato indipendente dalle unità lessicali circostanti. Questo significa che la probabilità di un token non dipende dal contesto delle parole precedenti o successive. In altre parole, il modello tratta ogni token come un'entità separata[14]. Il modello di linguaggio Unigram assume che la probabilità di un dato token sia determinata dalla sua frequenza nel corpus originale, divisa per la somma delle frequenze di tutti i token nel vocabolario.

Questo calcolo può essere rappresentato dalla seguente formula:

$$P(\text{token}) = \frac{\text{Frequenza del token nel corpus}}{\sum \text{Frequenze di tutti i token nel vocabolario}}$$

In questa formula, P(token) rappresenta la probabilità del token, mentre la frequenza del token nel corpus è il numero di volte in cui appare nel testo originale. La somma delle frequenze di tutti i token nel vocabolario garantisce che le probabilità di tutti i token nel vocabolario sommino a 1, creando una distribuzione di probabilità valida.

- **Creazione del vocabolario**: Nella fase di creazione del vocabolario del tokenizzatore Unigram, si inizia con la formazione di un ampio vocabolario iniziale. Questo vocabolario viene generato esaminando il corpus di testo e identificando una varietà di sotto-segmenti. Tali segmenti possono includere parole intere, loro frammenti, e n-grammi di caratteri, che possono essere ottenuti direttamente dal testo o attraverso l'applicazione di algoritmi come il Byte Pair Encoding (BPE). Ogni sotto-parola identificata viene poi assegnata una probabilità specifica basata sulla sua frequenza di occorrenza nel corpus. La probabilità di ciascuna sotto-parola riflette la sua rilevanza e rappresentatività nel contesto globale del testo, fornendo una base solida per il successivo processo di tokenizzazione e ottimizzazione del vocabolario.

- **Ottimizzazione del vocabolario**: Nell'ottimizzazione del tokenizzatore Unigram Language Model, il processo si concentra sul calcolo e la minimizzazione di una funzione di perdita che valuta l'efficacia del vocabolario corrente nel rappresentare il corpus di testo. La funzione di perdita è tipicamente formulata come la somma delle log likelihood negative delle parole

[14]Hugging Face. *Unigram tokenization.* URL: `https://huggingface.co/learn/nlp-course/chapter6/7?fw=pt`.

nel corpus, espressa matematicamente come $L = -\sum_{w \in Corpus} \log P(w)$, dove $P(w)$ indica la probabilità di ciascuna parola w nel corpus secondo il modello Unigram. Questo calcolo della perdita serve come base per valutare quali token, o sotto-parole, sono meno cruciali per la rappresentazione precisa del corpus. L'algoritmo procede alla valutazione dell'incremento della perdita che deriverebbe dalla rimozione di ciascun token. I token che causano il minor aumento della perdita sono considerati i meno importanti e quindi i migliori candidati per la rimozione. Tuttavia, invece di rimuovere singolarmente i token, l'algoritmo elimina una percentuale predefinita di token che mostrano il minimo incremento di perdita, un approccio noto come rimozione in batch. Questa procedura di rimozione in batch è ripetuta iterativamente fino a raggiungere la dimensione desiderata del vocabolario. In questo modo, il modello Unigram ottimizza il vocabolario, riducendo la sua dimensione pur mantenendo le caratteristiche linguistiche essenziali per rappresentare accuratamente il corpus di testo.

- **Tokenizzazione del testo**: Nella tokenizzazione del testo tramite il tokenizzatore Unigram Language Model, una volta che il vocabolario è stato ottimizzato, l'algoritmo di Viterbi viene utilizzato per analizzare e suddividere il testo in token. Questo processo inizia con l'identificazione di tutte le possibili sequenze di sotto-parole (o token) che possono comporre il testo, basandosi sulle probabilità di ogni sotto-parola nel vocabolario ottimizzato. L'algoritmo di Viterbi poi calcola la probabilità complessiva per ciascuna di queste possibili sequenze di sotto-parole. L'obiettivo è identificare la sequenza che ha la maggior probabilità di rappresentare correttamente il testo, considerando le probabilità individuali dei token e la loro combinazione nel contesto del testo. La sequenza selezionata dall'algoritmo di Viterbi è quella che meglio si allinea con la struttura e il significato del testo originale, sfruttando la distribuzione e la frequenza delle sotto-parole come apprese durante il processo di ottimizzazione del vocabolario.

Vantaggi del Tokenizzatore Unigram LM

- Adattabilità linguistica efficace per lingue con strutture morfologiche complesse e senza spazi bianchi.

- Gestione efficiente delle parole fuori dal vocabolario (OOV) attraverso la scomposizione in sotto-unità note.
- Flessibilità nella tokenizzazione basata sulle frequenze delle sotto-parole nel corpus.
- Utilità in contesti multilingue, dove un unico modello può essere addestrato su diversi linguaggi.

Svantaggi del Tokenizzatore Unigram LM

- Dipendenza dalla qualità e rappresentatività del corpus di addestramento.
- Complessità computazionale elevata nel processo di ottimizzazione del vocabolario.
- Mancanza di contestualizzazione delle parole, limitando l'efficacia in alcuni scenari di NLP.
- Rischio di overfitting se addestrato su un corpus di testo molto specifico.

2.5 Sentencepiece

SentencePiece, creato da Taku Kudo e John Richardson di Google[15], è stato introdotto nel 2018 come soluzione innovativa per il tokenizzatore e detokenizzatore di sotto-parole. Progettato principalmente per applicazioni di elaborazione del linguaggio naturale (NLP), SentencePiece si distingue per la sua capacità di lavorare direttamente con frasi grezze, senza la necessità di una pre-elaborazione del testo. Questa caratteristica lo rende particolarmente adatto per lingue che non hanno una chiara segmentazione delle parole, come il giapponese o il cinese. L'ampia applicabilità di SentencePiece lo ha reso uno strumento prezioso in diversi modelli di linguaggio basati su reti neurali, specialmente in quelli focalizzati sulla traduzione automatica e generazione di testo. Tra questi, SentencePiece è utilizzato in modelli di grande scala come BERT e GPT di OpenAI, dove facilita la gestione efficiente di grandi quantità di testo in diverse lingue. Inoltre, SentencePiece è impiegato anche in XLNet, un modello di linguaggio trasformativo di nuova generazione. XLNet

[15]Taku Kudo e John Richardson. ≪SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing≫. In: *Google, Inc.* (2018).

si distingue per il suo approccio unico nell'apprendimento del linguaggio, combinando i vantaggi dei modelli autoregressivi (come GPT) e autoencoder (come BERT). L'uso di SentencePiece in XLNet è particolarmente vantaggioso perché permette di gestire in modo efficiente e flessibile la varietà linguistica e la complessità del testo, contribuendo significativamente alla sua capacità di comprendere e generare linguaggio naturale in modo più accurato e versatile.

2.5.1 Workflow

SentencePiece si distingue nel panorama della tokenizzazione del testo per il suo approccio unico e metodico.

- **Pre-Processing**: All'inizio SentencePiece, affronta la normalizzazione del testo, un processo fondamentale che coinvolge la conversione dei caratteri Unicode in una forma standardizzata. Questo passaggio assicura la consistenza del testo, affrontando le variazioni come le differenze tra maiuscole e minuscole, oltre ad altre anomalie di codifica. Tale normalizzazione è essenziale per garantire che il testo sia omogeneo e pronto per la successiva fase di tokenizzazione. Un elemento distintivo di SentencePiece è il suo trattamento degli spazi bianchi. Invece di ignorarli o trattarli come separatori impliciti di parole, SentencePiece li converte in un simbolo visibile e distinto, come "". Questa sostituzione trasforma gli spazi in elementi tangibili del testo, allineandoli agli altri caratteri. Questo passaggio è particolarmente cruciale per la corretta tokenizzazione in lingue che non utilizzano gli spazi per delimitare chiaramente le parole, come il giapponese o il cinese. Allo stesso tempo, migliora la tokenizzazione anche in lingue con spazi, poiché consente di catturare più fedelmente la struttura e le peculiarità del testo.

- **Costruzione del vocabolario**: Nel processo di funzionamento di SentencePiece, la generazione del vocabolario è un passaggio fondamentale che determina la modalità di scomposizione e tokenizzazione del testo. SentencePiece offre due metodi per la creazione del vocabolario: Byte Pair Encoding (BPE) e Unigram Language Model, ciascuno adatto a diversi contesti e tipi di lingua. Il metodo BPE viene solitamente utilizzato in situazioni dove

la lingua ha una morfologia complessa o ricca, come nel caso di lingue con molte forme flesse per le stesse parole. BPE è efficace nel gestire questa varietà, identificando sotto-parole comuni nelle diverse forme. Questo metodo è particolarmente utile nelle applicazioni di traduzione automatica, dove la comprensione dettagliata della struttura delle parole è essenziale. D'altra parte, il modello Unigram è più adatto per contesti con una vasta gamma di distribuzione del testo o per corpora di grandi dimensioni e varietà linguistiche. Il modello Unigram adotta un approccio probabilistico, valutando la probabilità di ogni sotto-parola, il che lo rende più flessibile in ambienti dove le strutture morfologiche non sono altrettanto complesse o dove il vocabolario deve adattarsi dinamicamente a diverse distribuzioni di testo.

- **Gestione di OOV**: SentencePiece gestisce le Out-Of-Vocabulary (OOV) words, ovvero parole non presenti nel suo vocabolario predefinito, attraverso un approccio che sfrutta la sua capacità di tokenizzare il testo a un livello più granulare. Invece di affidarsi a un elenco fisso di parole intere, SentencePiece decompone il testo in sotto-parole o addirittura in singoli caratteri, a seconda della necessità. Quando SentencePiece incontra una parola che non è presente nel suo vocabolario, la spezza in sotto-unità più piccole che sono presenti nel vocabolario. Ad esempio, una parola OOV può essere suddivisa in più sotto-parole o, nel caso in cui anche le sotto-parole non siano nel vocabolario, in singoli caratteri. Questo approccio assicura che SentencePiece possa gestire qualsiasi parola, anche quelle mai incontrate durante la fase di addestramento del vocabolario.

- **Encoder e Decoder**: Nel funzionamento di SentencePiece, l'encoder e il decoder sono integrati strettamente nel flusso di lavoro di tokenizzazione e detokenizzazione. Dopo che il testo è stato normalizzato e gli spazi bianchi sono stati adeguatamente trattati, entra in gioco l'encoder. La sua funzione è di prendere questo testo preparato e scomporlo in un insieme di sotto-parole o simboli basati sul vocabolario generato, che può derivare sia dal metodo Byte Pair Encoding (BPE) sia dal Unigram Language Model. Questi elementi del testo vengono quindi convertiti in ID univoci, creando una sequenza di token che rappresenta il testo originale in una forma che può essere facilmente elaborata da modelli di machine learning.

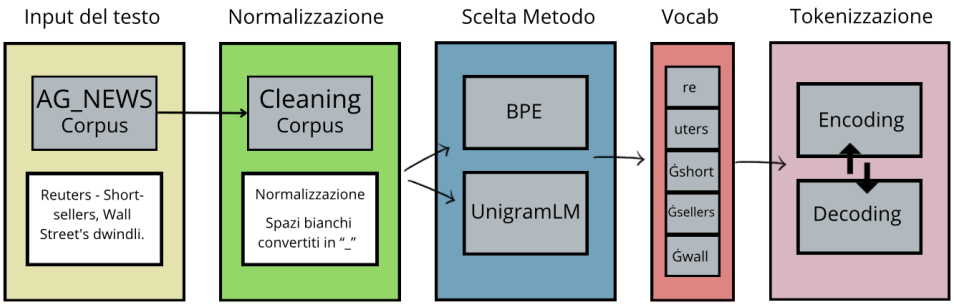

Figura 2.3: Workflow di SentencePiece

Successivamente, quando è necessario ricostruire il testo originale, ad esempio dopo un processo di traduzione automatica o al termine di un'analisi di NLP, il decoder entra in azione. Prende la sequenza di token o ID forniti dall'encoder e li converte di nuovo nelle sotto-parole corrispondenti, utilizzando il vocabolario di SentencePiece. Questo processo include la reinserzione degli spazi bianchi al posto del simbolo speciale usato durante la normalizzazione. In questo modo, il decoder riesce a ripristinare il testo nella sua forma leggibile e originale. Così, l'encoder e il decoder in SentencePiece lavorano in tandem per garantire che il testo passi fluidamente dalla sua forma naturale a una tokenizzata e viceversa, mantenendo l'integrità e il significato originale durante tutto il processo.

2.5.2 Vantaggi e Svantaggi

Vantaggi di SentencePiece

- **Indipendenza dalla Lingua**: SentencePiece non richiede un linguaggio specifico o regole di pre-elaborazione, rendendolo utilizzabile con una vasta gamma di lingue, comprese quelle con strutture morfologiche complesse o senza spazi bianchi.
- **Gestione delle Parole Fuori dal Vocabolario (OOV)**: Grazie alla sua capacità di scomporre il testo in sotto-parole o persino in singoli caratteri, SentencePiece gestisce efficacemente le parole OOV, riducendo gli errori di tokenizzazione.

- **Flessibilità nel Tokenizzare**: Offre due approcci di tokenizzazione (BPE e Unigram), permettendo agli utenti di scegliere il metodo più adatto in base alla natura del loro corpus di testo.
- **Efficienza in Contesti Multilingue**: È particolarmente utile in contesti multilingue e per la costruzione di modelli di machine learning che richiedono un unico sistema di tokenizzazione per diversi linguaggi.
- **Miglioramento della Coerenza e della Precisione**: Migliora la coerenza e la precisione in applicazioni come la traduzione automatica e la generazione di testo, poiché non dipende dalla correttezza della pre-tokenizzazione.

Svantaggi di SentencePiece

- **Complessità nel Gestire Vocabolari Grandi**: La gestione di vocabolari molto grandi può diventare complessa e meno efficiente, specialmente quando si utilizza il modello Unigram con un grande numero di sotto-parole.
- **Mancanza di Sensibilità al Contesto**: SentencePiece tokenizza basandosi solo sulla frequenza delle sotto-parole, senza considerare il contesto in cui appaiono, il che può essere un limite in alcune applicazioni di NLP.
- **Overfitting con Testi Specifici**: C'è il rischio di overfitting se il modello è addestrato su un corpus di testo molto specifico, rendendo il sistema meno adattabile a testi diversi da quelli di addestramento.
- **Prestazioni Varie con Diverse Lingue**: Anche se è indipendente dalla lingua, le prestazioni possono variare significativamente a seconda della lingua e della natura del testo, con alcune lingue che beneficiano di più della sua metodologia rispetto ad altre.
- **Risorse Computazionali per l'Addestramento**: L'addestramento di un modello SentencePiece, specialmente con un corpus di testo molto grande, può richiedere risorse computazionali significative.

3. Capitolo 3

Lo studio condotto si focalizza sul confronto di vari tokenizzatori, includendo sia quelli da addestrare che quelli pre-addestrati utilizzati nei Large Language Models (LLM), con un'attenzione particolare ai tokenizzatori a livello morfologico. La nostra domanda di ricerca è formulata come segue:

RQ-1: Qual è l'impatto dei diversi metodi di tokenizzazione sulla performance della modellazione della lingua inglese in compiti di classificazione delle notizie?

Per raggiungere questo scopo, sono stati selezionati e addestrati tre modelli di machine learning: Naive Bayes, Decision Tree e Random Forest. Questi modelli sono stati impiegati per esaminare l'influenza dei diversi approcci di tokenizzazione nel contesto specifico del task di classificazione delle notizie. La lingua presa in considerazione per lo studio è l'inglese, e il dataset utilizzato per l'addestramento e la validazione dei modelli è AG_NEWS, una fonte ben nota nel campo della classificazione delle notizie.
Il dataset in esame è strutturato in tre colonne fondamentali: la prima colonna rappresenta l'indice della classe, la quale serve a identificare il tipo di notizia; la seconda contiene il titolo della notizia; e la terza fornisce una descrizione dettagliata della stessa. Dalla Figura 3.1, è evidente che le varie classi di notizie sono distribuite in maniera equilibrata. Tale equilibrio può suggerire l'impiego di tecniche di oversampling o undersampling, prassi comuni nei dataset disponibili su piattaforme come Kaggle, dove il dataset è stato acquisito.

Indice della Classe	**Categoria**	**Frequenza**
1	World	30000
2	Sports	30000
3	Business	30000
4	Sci/Tech	30000

Tabella 3.1: Distribuzione delle Classi delle Categorie di notizie

Nonostante ciò, si è presa la decisione di non apportare modifiche alle probabilità a priori e a posteriori del modello. Questa scelta si basa sull'ipotesi che il bilanciamento osservato nelle classi possa, in realtà, riflettere la distribuzione effettiva delle notizie nel mondo reale. Pertanto, mantenere le proporzioni originali delle classi potrebbe fornire una rappresentazione più fedele e realistica delle notizie, contribuendo così a una migliore interpretazione e analisi dei dati.

3.1 Modus Operandi

Il processo di tokenizzazione, illustrato nei primi quattro rettangoli nella figura, si sviluppa attraverso tre fasi distinte e sequenziali. La prima fase prevede una pulizia del testo. Questa operazione è cruciale per garantire l'integrità dei dati e comporta l'eliminazione dei caratteri non validi o irrilevanti, assicurando così un'analisi testuale priva di distorsioni.

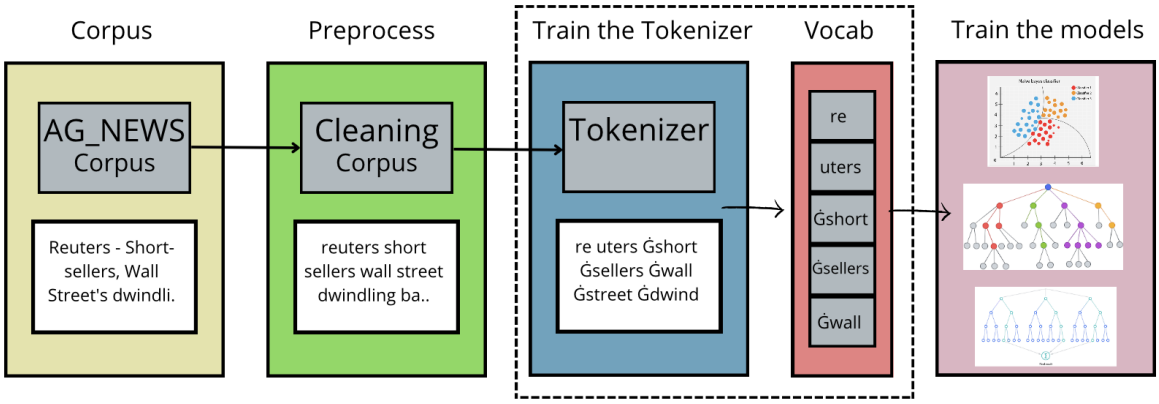

Figura 3.1: Modus Operandi

Successivamente, si procede con la fase di addestramento del tokenizer. Questa operazione è intrinsecamente legata alla scelta di una dimensione di vocabolario ben definita e prestabilita, la quale serve da fondamento per la costruzione del modello linguistico. È importante sottolineare che tale fase non si applica ai tokenizer già pre-addestrati, i quali possiedono un set di vocaboli e regole predefiniti basati su corpora di testo estesi. L'ultima tappa di questo processo comporta

l'elaborazione del corpus attraverso l'impiego del tokenizer addestrato. Questa fase è essenziale per la trasformazione dei dati testuali in un formato che sia compatibile con le esigenze dei modelli di machine learning. Dopo aver completato il processo di tokenizzazione, si procede con l'addestramento di tre specifici modelli di machine learning sui dati tokenizzati. Per quanto concerne la rappresentazione in embedding, adottiamo il metodo TF-IDF (Term Frequency-Inverse Document Frequency). Tale metodologia permette di convertire il testo in una rappresentazione numerica, attribuendo un peso alle parole in base alla loro frequenza all'interno dei documenti e alla loro distribuzione nell'intero dataset. Concludendo, si confrontano i risultati ottenuti in termini di accuratezza, precisione, recall e F-score. Questo confronto è essenziale per garantire che le prestazioni dei modelli rispecchino accuratamente le aspettative e gli standard attuali nel campo della classificazione di testi basata su machine learning.

3.2 Analisi dei risultati

3.2.1 Confronto dei Tokenizzatori Basato sul Tempo di Addestramento e Velocità di Tokenizzazione

Nella selezione di un tokenizzatore per progetti di machine learning e NLP, la velocità e il tempo di addestramento sono cruciali. Un tokenizzatore veloce è essenziale per processare grandi volumi di dati efficacemente e per garantire prestazioni in tempo reale in applicazioni come assistenti vocali o sistemi di traduzione. Allo stesso tempo, ridurre il tempo di addestramento è importante per limitare i costi e accelerare lo sviluppo del progetto. Per confrontare i tokenizzatori, sono stati eseguiti test su un MacBook Air con processore M1. I dataset utilizzati per questi test provengono da ag_news, descritto precendentemente composto da:

- train_set di 120.000 righe e 3 colonne, con un peso complessivo di 29 MB
- test_set di test di 7.600 righe e 3 colonne, del peso di 1,8 MB
- i titoli di addestramento dal peso di 5,2 MB

Tokenizzatore	Tempo di Addestramento	Velocità di tokenizzazione
WHITE SPACE	Non necessita di addestramento	9.31 ms
BPE	Medio	20 m
BPE ROBERTA	Pre-addestrato	5.45 s
MORFESSOR	Alto	over 50 m
MOSES	Non necessita di addestramento	4.83 s
SENTENCEPIECE XLN	Pre-addestrato	7.79 s

Tabella 3.2: Confronto dei tempi di addestramento e delle velocità di tokenizzazione per diversi tokenizzatori.

Dai risultati del test visibili nella tabella si evincono le seguenti osservazioni:

- Il tokenizzatore White Space non necessita di un processo di addestramento specifico, in quanto opera sulla base di regole predefinite che segmentano il testo utilizzando gli spazi vuoti come delimitatori. Tale approccio è simile a quello adottato dal tokenizzatore Moses, il quale si basa su regole morfologiche per la segmentazione del testo.

- I tokenizzatori BPE con Roberta e SentencePiece con XLN, sono di strumenti già preaddestrati su vasti corpora di testo, offrendo il vantaggio di essere immediatamente operativi e performanti in termini di tempo di tokenizzazione. La loro efficienza deriva dall'essere stati ottimizzati durante il processo di preaddestramento. Tuttavia, un possibile svantaggio di questi tokenizzatori preaddestrati risiede nel loro corpus di riferimento, che potrebbe non essere perfettamente allineato con i requisiti specifici di un nuovo compito di tokenizzazione, necessitando così di un addestramento ulteriore e più mirato.

- Morfessor e BPE richiedono un addestramento diretto su un corpus fornito in input. Per i test effettuati, sono stati selezionati i titoli del set di addestramento, con una dimensione complessiva di 5,2 MB. Si è osservato che il tokenizzatore BPE ha completato l'addestramento con successo in un intervallo di tempo che varia tra i 20 e i 50 minuti. Morfessor, invece, non ha concluso il processo di addestramento entro il limite di 50 minuti impostato.

Per quanto riguarda il tempo di tokenizzazione, BPE ha richiesto un periodo simile, tra i 20 e i 50 minuti, per elaborare completamente i dati. Questa prestazione può essere attribuita a una possibile inefficienza nell'implementazione del codice, la quale potrebbe influenzare negativamente le prestazioni di tokenizzazione.

I risultati ottenuti differenziati per task sono riportati nella figura 3.2

Figura 3.2: Confronto dei Tokenizzatori Basato sul Tempo di Addestramento e Velocità di Tokenizzazione

3.2.2 Analisi Comparativa in termini di memoria in addestramento e tokeizzazione

Un criterio efficace per confrontare i diversi tokenizzatori nel campo dell'elaborazione del linguaggio naturale riguarda l'analisi della memoria computazionale impiegata. Questo fattore è fondamentale per valutare l'efficienza e l'adeguatezza di un tokenizzatore per specifici compiti o contesti operativi. Nel contesto dell'analisi complessa del codice, il termine "memoria" assume un significato specifico e multidimensionale, essenziale per comprendere come il codice viene elaborato ed eseguito. La memoria, in questo ambito, si riferisce allo spazio di archiviazione utilizzato per conservare i dati e le istruzioni necessarie per l'esecuzione di un programma. La sua gestione è cruciale per le prestazioni e l'efficienza del codice. In questo contesto pratico la memoria utilizzata viene osservata su diversi fronti:

- **Fase di Addestramento**: Durante l'addestramento, il tokenizzatore impara a identificare e a suddividere il testo in token. Questo processo richiede la memorizzazione di grandi quantità

di dati testuali e la gestione delle informazioni sui token e sulle loro relazioni.

- **Fase di Applicazione della Tokenizzazione**: In questa fase, il tokenizzatore utilizza la memoria per analizzare nuovi testi, scomporli in token e recuperare le informazioni corrispondenti dal vocabolario.

Il tokenizzatore basato su spazi bianchi rappresenta la semplicità estrema. Non richiede quasi alcuna memoria per l'addestramento perché si limita a dividere il testo in token basandosi sugli spazi. Al contrario, il Byte Pair Encoding (BPE) personalizzato segue un approccio più sofisticato. L'addestramento su un corpus di testo specifico richiede un considerevole uso della memoria. Questo è dovuto al processo iterativo in cui il tokenizzatore identifica e fonde le coppie di caratteri o byte più frequenti per formare nuovi token. Durante la tokenizzazione, il BPE richiede un uso moderato delle risorse, poiché si affida a un vocabolario preaddestrato e ottimizzato. Il BPE pre-addestrato, come quello usato in Roberta, non impatta sulle risorse computazionali durante la fase di addestramento, in quanto è stato già addestrato e ottimizzato precedentemente. Durante la tokenizzazione, utilizza risorse simili al BPE personalizzato, sfruttando il vantaggio di un vocabolario già ottimizzato. Moses, che non include una fase di addestramento, riduce significativamente l'uso della memoria durante la sua implementazione. Si basa su un insieme di regole fisse per la tokenizzazione.

Tokenizzatore	Memoria in Addestramento	Memoria in Tokenizz.
Spazi Bianchi	Minima	759.33 MiB
BPE	Alta	-
BPE Pre-Addestrato	Pre-Addestrato	592.55 MiB
Moses	Minima	457.47 MiB
Morfessor	Alta	-
SentencePiece (XLNet)	Pre-Addestrato	728.08 MiB

Tabella 3.3: Analisi Comparativa in termini di memoria

Morfessor, focalizzato sulla segmentazione morfologica, richiede una notevole quantità di memoria durante l'addestramento per analizzare e comprendere la struttura morfologica delle parole. SentencePiece, come utilizzato in XLNet, essendo pre-addestrato, non richiede risorse aggiuntive durante l'addestramento nel contesto attuale. Il suo processo di tokenizzazione è efficiente in termini di risorse, beneficiando di un vocabolario già ottimizzato per una varietà di contesti linguistici.

I risultati ottenuti differenziati per task sono riportati nella figura **??**

Figura 3.3: Analisi Comparativa in termini di memoria in addestramento e tokenizzazione

3.2.3 Analisi Comparativa in termini di Dimensione del Vocabolario e Output di Tokenizzazione

Un aspetto importante da considerare quando si valutano le prestazioni dei tokenizzatori è l'output della tokenizzazione. Quando si elabora un nuovo testo, il tokenizzatore lo suddivide in una serie di token utilizzando come guida il vocabolario preesistente. Ogni parola, frase o simbolo nel testo viene confrontato con le voci del vocabolario e viene sostituito con un identificativo specifico associato a quel token.

Di conseguenza, l'output della tokenizzazione è costituito da una sequenza di questi identificativi di token. Questa sequenza rappresenta il testo originale in modo accurato, ma è formattata in un modo che rende più agevole il successivo processo di elaborazione dei dati. In altre parole, la tokenizzazione trasforma il testo in una forma strutturata che può essere facilmente utilizzata nei passaggi successivi del pre-processing.

- **White Spaces**: Questo metodo semplicemente conta gli spazi bianchi per dividere le parole, (nell'esempio 3.4 si ottengono 13 token, quantità più piccola rispetto agli altri tokenizzatori). È il metodo più semplice e diretto, ma non tiene conto di punteggiatura o strutture complesse del linguaggio.

Tokenizzatori	Testo	N° Token
White Spaces	unions representing workers turner newall say disappointed talks stricken parent firm federal mogul	13
BPE	un ions Ġre p res ent ing Ġw ork ers Ġt urn er Ġnew all Ġs ay Ġd is app o int ed Ġtalks Ġst ric k en Ġp ar ent Ġf irm Ġf ed er al Ġm og ul	40
BPE Roberta	un ions Ġrepresenting Ġworkers Ġturn er Ġnew all Ġsay Ġdisappointed Ġtalks Ġstricken Ġparent Ġfirm Ġfederal Ġmogul'	16
XLN	6432 4471 1042 808 118 109 2225 248 6869 916 17 17766 4090 1338 819 28516 4 3	18
Moses	unions representing workers turner newall say disappointed talks stricken parent firm federal mogul	13

Tabella 3.4: Output della Tokenizzazione dei Testi

- **BPE**: L'output generato dal BPE è influenzato dal vocabolario sviluppato nella fase di allenamento. Utilizzando un vocabolario personalizzato, si tende ad ottenere un numero maggiore di token rispetto ad altri metodi di tokenizzazione. Questo avviene perché, durante l'addestramento, il corpus fornito al BPE per la creazione del vocabolario e la quantità di termini da includere (un parametro da regolare) dovrebbero essere proporzionali alla mole dei dati di training. In caso contrario, si rischia una tokenizzazione delle parole a livello di singoli caratteri come nel caso in esempio con un numero di token pari a 40.

- **BPE Roberta**: Per il BPE pre-addestrato utilizzato in RoBERTa, non si incontrano problemi

di tokenizzazione eccessivamente frammentata, come avviene nel BPE preaddestrato. Questo perché RoBERTa è stato addestrato in maniera più efficiente e ottimizzata, seguendo tutte le regole e le migliori pratiche necessarie per il BPE. Di conseguenza, il modello di RoBERTa riesce a catturare l'essenza vera del BPE, offrendo una tokenizzazione che rispecchia in modo più fedele la struttura linguistica e semantica del testo analizzato.

- **XLN**: Utilizzando il tokenizzatore SentencePiece, fornisce un output di lunghezza simile a quello ottenuto dal BPE, ma con una differenza sostanziale: XLNet impiega una rappresentazione numerica fin dall'inizio. Come nel caso di RoBERTa, il tokenizzatore SentencePiece impiegato da XLNet è pre-addestrato. Questo significa che è stato ottimizzato su un ampio corpus di testo per catturare efficacemente le peculiarità linguistiche. Grazie a questo addestramento approfondito e specifico, XLNet sfrutta appieno le potenzialità del tokenizzatore SentencePiece, riuscendo a coglierne la vera essenza e a fornire una tokenizzazione che sia sia efficiente sia rappresentativa delle strutture linguistiche complesse.

- **Moses**: simile al metodo White Spaces, ha generato 13 token nel medesimo esempio. Per la lingua inglese, Moses impiega regole di tokenizzazione che in molti casi sono paragonabili a quelle di White Spaces, ma con alcune ottimizzazioni specifiche per la traduzione.

I risultati ottenuti differenziati per task sono riportati nella figura 3.4

Figura 3.4: Analisi Comparativa in termini di Dimensione del Vocabolario e Output di Tokenizzazione

3.2.4 Analisi Comparativa dei Tokenizzatori in relazione all'accurancy nei Modelli di Machine Learning

Nella tabella 3.5 si mostrano i risultati ottenuti da tre diversi modelli di machine learning impiegati per classificare le notizie in base alla loro accuratezza. Ognuno di questi modelli è stato addestrato sullo stesso insieme di dati, ma con diverse tecniche di tokenizzazione. Per rappresentare i token nello spazio delle caratteristiche, è stato adottato il metodo TF-IDF.

	Naive bayes	**Decision Tree**	**Random forest**
White space	0.89	0.81	0.89
BPE	0.80	0.70	0.83
BPE Roberta	0.89	0.79	0.88
XLN	0.88	0.77	0.89
Moses	0.89	0.81	0.89

Tabella 3.5: Confronto delle prestazioni dei modelli su diversi tipi di tokenizzatori.

Naive Bayes

La relazione tra l'efficacia dell'algoritmo Naive Bayes e la scelta del tokenizzatore nel campo dell'elaborazione del linguaggio naturale offre uno spaccato di come la struttura dei dati influenzi profondamente la performance dei modelli di machine learning. Nel cuore di questa dinamica c'è l'interazione tra la natura dei tokenizzatori e l'assunzione fondamentale di indipendenza delle caratteristiche su cui si basa Naive Bayes. Naive Bayes, un modello probabilistico, dipende dall'ipotesi che le caratteristiche (in questo caso, i token del testo) siano indipendenti l'una dall'altra. Questa assunzione semplifica notevolmente i calcoli, ma può anche essere una fonte di debolezza se le caratteristiche reali non sono effettivamente indipendenti. Ecco dove entra in gioco il ruolo cruciale dei tokenizzatori. Il tokenizzatore basato su spazi bianchi, che ha mostrato elevate prestazioni con Naive Bayes, ha un output di tokenizzazione più esiguo rispetto agli altri, risultando in un'esigua diversità di caratteristiche. Questa diversità può avvicinarsi all'ideale di indipendenza delle caratteristiche assunto da Naive Bayes. Nonostante ogni token possa non essere veramente

indipendente in un senso stretto, la vasta gamma di token unici riduce le interdipendenze potenziali, allineandosi meglio con le presupposizioni del modello. Per quanto riguarda il tokenizzatore BPE, addestrato manualmente, si registra un'accuratezza leggermente inferiore, pari a 0.80. Tale discrepanza potrebbe derivare dall'utilizzo di un set di addestramento di dimensioni ridotte, che ha condotto alla formazione di un output di tokenizzazione molto numeroso allontanandosi così dall'ipotesi di indipendenza. Tuttavia, l'alta efficienza di Naive Bayes con tokenizzatori avanzati come BPE Roberta e XLNet rivela una sfumatura interessante. Anche se questi tokenizzatori aumentano anch'essi la dimensione del output di tokenizzazione, sembra che lo facciano in un modo che mantiene o addirittura rafforza l'indipendenza delle caratteristiche, forse attraverso una selezione più mirata e significativa dei token. Questo suggerisce che la qualità dei token, in termini di rappresentazione delle informazioni e pertinenza per il testo, può essere altrettanto critica quanto il loro numero.

Decision Tree

Nello studio in oggetto, si è proceduto all'esplorazione dell'effetto di diversi tokenizzatori - quali "Moses", "White space", "BPE", "BPE Roberta" e "XLN" - sull'efficacia di modelli di Decision Tree nell'ambito della classificazione delle notizie. Ciò è stato condotto senza procedere a una specifica ottimizzazione dei parametri di tali modelli, allo scopo di valutare l'impatto iniziale che differenti metodi di tokenizzazione possono avere sulle performance di classificazione. I parametri dei modelli Decision Tree sono stati mantenuti ai valori predefiniti di scikit-learn versione 1.0.2. Questi parametri predefiniti includono il criterio di divisione 'gini,' il metodo di divisione 'best,' una profondità massima dell'albero impostata su None, un numero minimo di campioni richiesti per suddividere un nodo interno pari a 2, un numero minimo di campioni richiesti per essere in una foglia pari a 1, l'utilizzo di tutte le feature disponibili durante la ricerca della migliore divisione e un parametro random_state impostato a 42. Il Decision Tree è un algoritmo di apprendimento supervisionato che cerca di suddividere il set di dati in base alle feature più discriminanti per raggiungere l'obiettivo di classificazione. Tra i tokenizzatori utilizzati, "Moses" e "White space" hanno ottenuto un'accuracy più elevata rispetto agli altri, probabilmente a causa della loro tokenizzazione significativa, a seguito di un output di tokenizzazione più esiguo, che ha reso più agevole

per il Decision Tree identificare le divisioni ottimali. Il Bpe Personalizzato è il tokenizzatore che fornisce la performance peggiore questo probabilmente a causa delle modalità di addestramento non efficienti e di consceguenza che hanno portato ad un ampio output non pienamnete significativo.

Random Forest

L'efficacia superiore dell'algoritmo di Random Forest con alcuni tokenizzatori rispetto ad altri può essere attribuita a una serie di fattori intrinseci alla natura dell'algoritmo e alla tipologia di dati trattati. Random Forest, essendo un modello ensemble, trae beneficio dalla combinazione di previsioni di molteplici alberi di decisione, riducendo così il rischio di overfitting e aumentando la robustezza del modello di fronte a variazioni nei dati di input. Il tokenizzatore "White space", che separa le parole basandosi sugli spazi, offre un approccio di tokenizzazione molto diretto. Questo potrebbe significare che, per il dataset in esame, la gran parte delle informazioni utili alla classificazione è contenuta nel singolo uso delle parole piuttosto che nella loro composizione o nella morfologia. I modelli di Random Forest possono trarre vantaggio da questa semplicità, poiché ogni nodo dell'albero può facilmente valutare la presenza o l'assenza di una parola specifica nel testo per fare una previsione. "Moses" è un tokenizzatore che è stato originariamente progettato per il pre-processamento nel campo della traduzione automatica e potrebbe essere ottimizzato per riconoscere frasi ed espressioni ricorrenti che sono particolarmente rilevanti per la comprensione del testo. Se il dataset di classificazione delle notizie contiene molte di queste espressioni standardizzate, "Moses" potrebbe essere in grado di catturare le relazioni linguistiche che sono importanti per la classificazione, fornendo così ai modelli delle feature informative. Nonostante la complessità dei tokenizzatori "BPE Roberta" e "XLN", essi risultano efficaci con l'algoritmo di Random Forest. "BPE Roberta", con la sua analisi dettagliata della struttura delle parole, fornisce un set di dati che Random Forest può utilizzare per identificare e sfruttare pattern complessi, che potrebbero essere sfuggiti a metodi più semplici. "XLN", specializzato nel trattare contesti più ampi, alimenta l'algoritmo con una serie di dati contestuali che Random Forest elabora per cogliere dipendenze che si estendono su ampi segmenti di testo. Questi tokenizzatori, quindi, offrono un set di feature ottimale che permette a Random Forest di sfruttare al meglio la sua capacità di modellare le

interazioni complesse tra le variabili, portando a un incremento delle prestazioni di classificazione.

Bibliografia

Face, Hugging. *Byte-Pair Encoding tokenization*. URL: https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt.

— *RoBERTa*. URL: https://huggingface.co/docs/transformers/model_doc/roberta.

— *Unigram tokenization*. URL: https://huggingface.co/learn/nlp-course/chapter6/7?fw=pt.

Koehn, Philipp. «Moses: Statistical Machine Translation System, User Manual and Code Guide». In: *University of Edinburgh* (2022).

Koehn, Philipp et al. «Moses: Open Source Toolkit for Statistical Machine Translation». In: *Association for Computa- tional Linguistics* (2007).

Kudo, Taku. «Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates». In: *Google, Inc.* (2018).

Kudo, Taku e John Richardson. «SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing». In: *Google, Inc.* (2018).

Mielke, Sabrina J. et al. «Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP». In: *BigScience Workshop Tokenization Working Group* (2021).

Singh, Awaldeep. «Hugging Face: Understanding tokenizers». In: *Medium* (2022).

Smit, Peter. «Morfessor 2.0: Toolkit for statistical morphological segmentation». In: *Department of Information and Computer Science, Aalto University* (2014).

TORAMAN, CAGRI et al. «Impact of Tokenization on Language Models: An Analysis for Turkish». In: *Aselsan Research Center, Turkey* (2022).

Virpioja, Sami, Peter Smit e Miko Kurimo. «Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline». In: *Department of Information and Computer Science, Aalto University* (2013).

Yang, Zhilin et al. «XLNet: Generalized Autoregressive Pretraining for Language Understanding». In: *CoRR* (2019). URL: http://arxiv.org/abs/1906.08237.

Zouhar, Vilém et al. «A Formal Perspective on Byte-Pair Encoding». In: *ETH Zürich* (2023).