



STUDIO SHODWE

Tokenizzatori in Azione:

Confronto e Applicazione per la
Classificazione di Notizie in Diverse Metriche



NLP e Tecnologia nell'Era dell'Informazione

Nell'era dell'informazione digitale, la comprensione dei testi assume una nuova dimensione quando si tratta di elaborarli attraverso sistemi informatici. Questo ambito, dove la linguistica incontra la tecnologia, richiede una trasformazione significativa dei dati testuali per renderli accessibili e utilizzabili dalle macchine. Al centro di questo processo rivoluzionario sta l'Elaborazione del Linguaggio Naturale (NLP), una disciplina che si posiziona all'intersezione tra la linguistica, l'informatica e l'intelligenza artificiale, e che rappresenta un ponte fondamentale tra il linguaggio umano e la macchina.

I Pilastri dell'Elaborazione del Linguaggio Naturale



Tokenizzazione

La tokenizzazione è la procedura mediante la quale un testo viene suddiviso in unità più piccole, denominate "token".



I token sono le unità atomiche nel processo di NLP, che **fungono da ponte** tra il linguaggio naturale e la rappresentazione numerica che i modelli possono elaborare

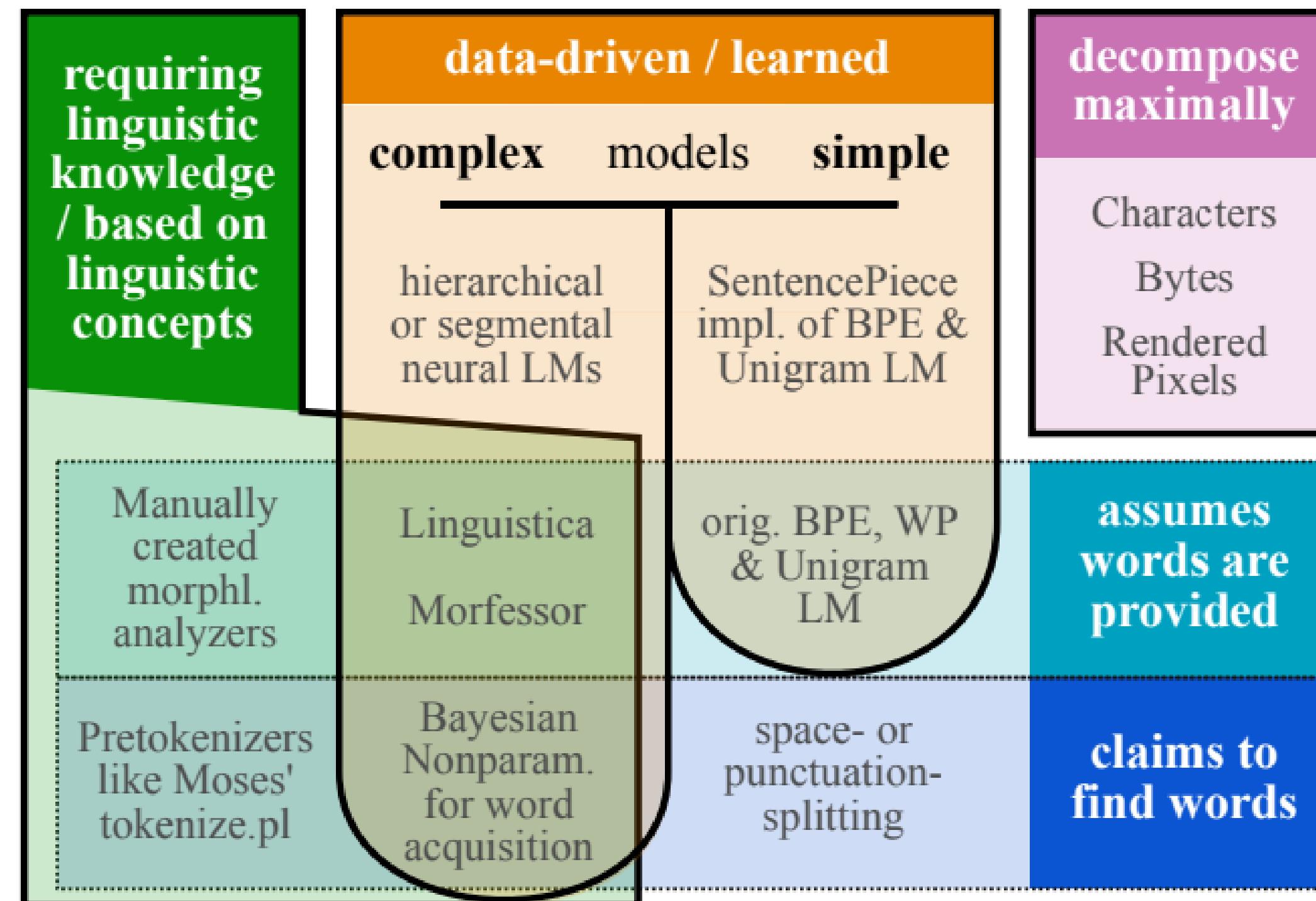


Una delle maggiori sfide nella tokenizzazione, oltre alla **complessità morfologica** delle lingue è la **gestione della semantica**. La semantica si riferisce al significato e all'interpretazione delle parole e delle frasi all'interno di un contesto specifico

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"]

Livelli di tokenizzazione

Metodologie di Tokenizzazione



BPE

Originariamente nato per la compressione dei dati da Storer e Szymanski nel 1982

Successivamente definito in maniera formale da Philip Gage nel 1994

- 1 Pre-tokenizzazione**
Nella fase iniziale, BPE richiede una pre-tokenizzazione del testo. Questo processo generalmente coinvolge la suddivisione del testo in unità di base come parole, spesso utilizzando separatori semplici come spazi e punteggiatura.
- 2 Inizio Addestramento**
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
- 3 Identificazioni coppie**
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 nuovo simbolo, che viene quindi aggiunto al vocabolario.
- 4 Criterio di Terminazione**
Il processo prosegue finché non viene soddisfatto un criterio di terminazione, che può essere un numero di iterazioni prefissato o una dimensione massima del vocabolario.

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

Schema iterazioni nella Tokenizzazione BPE

VANTAGGI

- **Riduzione della Complessità del Vocabolario**
- **Efficienza in Contesti Multilingue**
- **Gestione delle Parole Rare o Sconosciute**

SVANTAGGI

- **Mancanza di Contesto e Semantica**
- **Equilibrio nella Dimensione del Vocabolario**
- **Sfide nelle Lingue con Morfologia Complessa**

Implementazione in Python

Due Metodi differenti

BPE

Questa tecnica offre una personalizzazione flessibile, consentendo l'addestramento su qualsiasi insieme di dati e la libertà di selezionare la dimensione del vocabolario attraverso la scelta del numero di token.

BPE-ROBERTa

è una versione pre-addestrata che cattura l'essenza del BPE, eliminando la necessità di formazione e selezione manuale dei parametri, grazie a un'implementazione efficiente e ottimizzata del codice.

BPE Personalizzato

```
# pre-tokenize the corpus into words, BERT pre-tokenizer is used here
self.tokenizer = AutoTokenizer.from_pretrained("gpt2")
self.word_freqs = defaultdict(int)
self.splits = {}
self.merges = {}
```

← Pre-Tokenizzazione

```
# compute the frequencies of each word in the corpus
for text in self.corpus:
    words_with_offsets = self.tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_st
    new_words = [word for word, offset in words_with_offsets]
    for word in new_words:
        self.word_freqs[word] += 1

# compute the base vocabulary of all characters in the corpus
alphabet = []
for word in self.word_freqs.keys():
    for letter in word:
        if letter not in alphabet:
            alphabet.append(letter)
alphabet.sort()

# add the special token </w> at the beginning of the vocabulary
vocab = ["</w>"] + alphabet.copy()

# split each word into individual characters before training
self.splits = {word: [c for c in word] for word in self.word_freqs.keys()}
```

← Creazione
vocabolario e coppie
più frequenti

```
# merge the most frequent pair iteratively until the vocabulary size is reached
while len(vocab) < self.vocab_size:

    # compute the frequency of each pair
    pair_freqs = self.compute_pair_freqs()

    # find the most frequent pair
    best_pair = ""
    max_freq = None
    for pair, freq in pair_freqs.items():
        if max_freq is None or max_freq < freq:
            best_pair = pair
            max_freq = freq

    # merge the most frequent pair
    self.splits = self.merge_pair(*best_pair)
    self.merges[best_pair] = best_pair[0] + best_pair[1]
    vocab.append(best_pair[0] + best_pair[1])
return self.merges
```



Ottimizzazione
vocabolario in
diversi iter

BPE Roberta

RoBERTa (Robustly Optimized BERT Approach) è una versione avanzata del modello BERT, ottimizzata per una migliore performance in compiti di elaborazione del linguaggio naturale. Il tokenizer di RoBERTa utilizza un approccio BPE a livello di byte e non di carattere, che gli permette di gestire in modo efficace un'ampia varietà di lingue e caratteri speciali.

```
def __init__(self, model_name='roberta-base'):
    self.tokenizer = RobertaTokenizer.from_pretrained(model_name)

def process_text(self, text):
    # Tokenizza il testo
    tokens = self.tokenizer.tokenize(text)
    # Converte i token in una stringa
    return ' '.join(tokens)

def apply_pipeline(self, df, column_name):
    # Applica la pipeline di processamento del testo alla colonna specificata
    return df[column_name].apply(self.process_text)
```

MORFESSOR 2.0

Morfessor è una famiglia di metodi di apprendimento automatico probabilistico che trova segmentazioni morfologiche per parole, (**morfemi**) di una lingua, basandosi esclusivamente su dati di testo grezzo.



FLUSSO DI LAVORO

01

INIZIALIZZAZIONE: Il modello viene configurato con un lessico iniziale che comprende tutte le forme composte presenti nel set di addestramento.

02

TRAINING: Il modello elabora i dati di addestramento attraverso un processo iterativo. In ogni iterazione, tutte le forme composte presenti nel set di addestramento vengono esaminate.

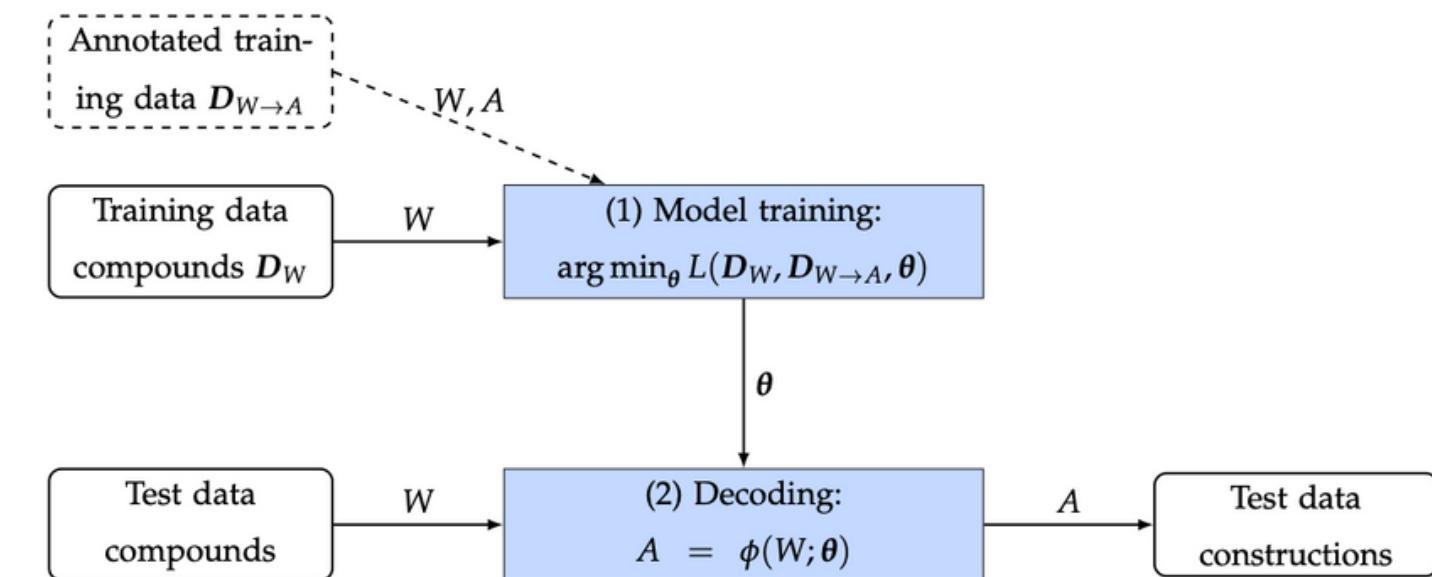
03

TRAINING: La selezione della segmentazione ottimale si basa sul calcolo del costo, che viene ottimizzato per convergere verso una soluzione efficiente.

04

DECODING: L'algoritmo di Viterbi esteso, nel modello ottimizzato, identifica segmentazioni ottimali in nuove forme, guidato dalla grammatica e dalle conoscenze acquisite in fase di addestramento.

Task	Compounds	Constructions	Atoms
Morphological segmentation	word forms	morphs	letters
Chinese word segmentation	sentences	words	letters
Chunking / shallow parsing	sentences	phrases	words



VANTAGGI

- Gestione delle Lingue Morfologicamente Ricche
- Apprendimento Automatico
- Riduzione della Dimensione del Vocabolario
- Supporto Semi-Supervisionato

SVANTAGGI

- Complessità per Lingue con Morfologia Semplice
- Dipendenza dal Corpus di Addestramento
- Complessità Computazionale
- Necessità di Regolazione dei Parametri

Implementazione in Python

```
# Creazione di un'istanza per I/O e del modello Morfessor
infile = path_del_file_titoli
io = morfessor.MorfessorIO()
train_data = list(io.read_corpus_file(infile))

# Creare un'istanza del modello Morfessor e caricare i dati
model = morfessor.BaselineModel()
model.load_data(train_data, count_modifier=log_func)

# Addestrare il modello
model.train_batch()
```

← Model Training

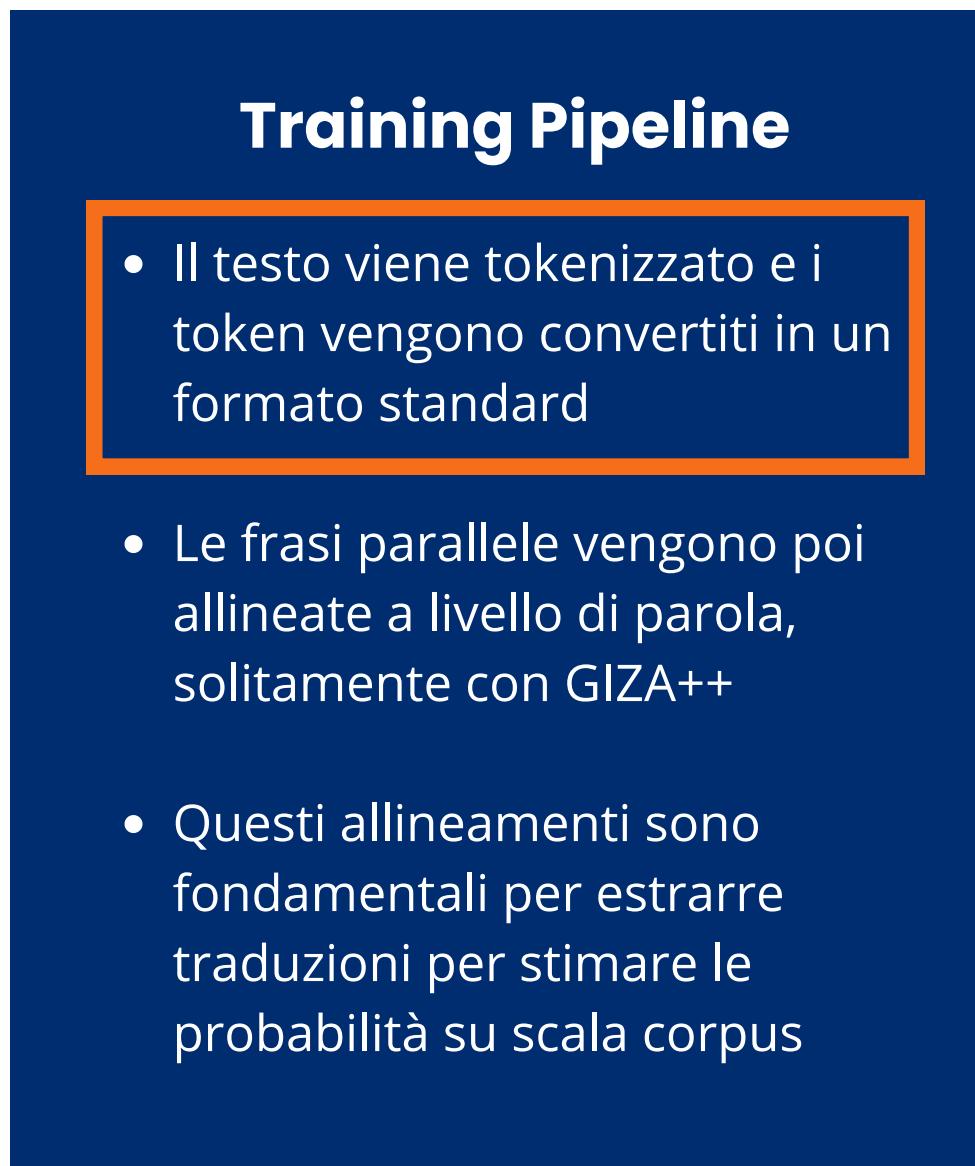
```
class MorfessorTokenizationPipeline:
    def __init__(self, model_path):
        self.model = morfessor.MorfessorIO().read_binary_model_file(model_path)

    def tokenize(self, text):
        return self.model.viterbi_segment(text)[0]
```

← Pipeline di tokenizzazione
con morfessor

MOSES

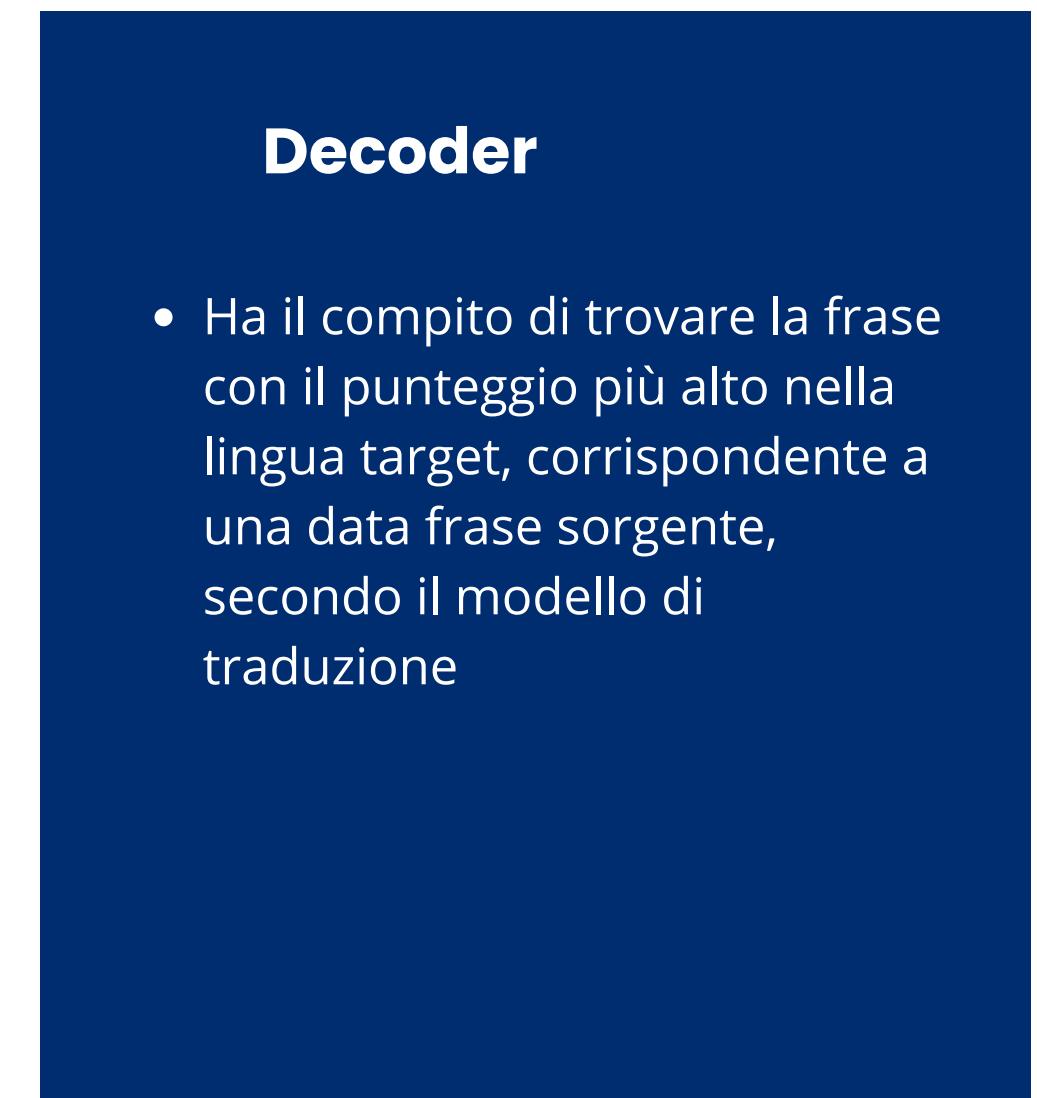
Moses rappresenta un'implementazione dell'approccio statistico alla traduzione automatica, utilizzato anche nei sistemi di traduzione online di Google e Microsoft, include tra i suoi toolkit un componente di tokenizzazione.



$$\begin{array}{c} \left(\begin{array}{c} je \\ PRO \\ je \\ 1st \end{array} \right) \left(\begin{array}{c} vous \\ PRO \\ vous \\ 1st \end{array} \right) \left(\begin{array}{c} achet \\ VB \\ acheter \\ 1st / present \end{array} \right) \left(\begin{array}{c} un \\ ART \\ un \\ masc \end{array} \right) \left(\begin{array}{c} chat \\ NN \\ chat \\ sing / masc \end{array} \right) \\ \downarrow \end{array}$$

$$\begin{array}{c} \left(\begin{array}{c} i \\ PRO \\ i \\ 1st \end{array} \right) \left(\begin{array}{c} buy \\ VB \\ tobuy \\ 1st / present \end{array} \right) \left(\begin{array}{c} you \\ PRO \\ you \\ 1st \end{array} \right) \left(\begin{array}{c} a \\ ART \\ a \\ sing \end{array} \right) \left(\begin{array}{c} cat \\ NN \\ cat \\ sing \end{array} \right) \end{array}$$

Figure 2. Factored translation



MOSES TOKENIZER

01

Utilizzo del Wrapper

Python: viene utilizzato uno script Perl denominato tokenizer.perl per eseguire il processo di tokenizzazione

02

Specificazione della Lingua: questo permette al Moses Tokenizer di applicare le regole di tokenizzazione appropriate per la lingua specificata

03

Regole di Tokenizzazione: sono basate su spazi, punteggiatura e altre convenzioni linguistiche. Queste regole variano da lingua a lingua e vengono applicate in modo coerente durante il processo.

04

Output Tokenizzato: Il testo subisce la tokenizzazione e l'output consiste in una versione tokenizzata del testo di input. Ogni token è separato da spazi, creando una struttura standardizzata.

REGOLE PER LINGUA

nonbreaking_prefix.hu

nonbreaking_prefix.is

nonbreaking_prefix.it

nonbreaking_prefix.kn

nonbreaking_prefix.lt

nonbreaking_prefix.lv

nonbreaking_prefix.ml

nonbreaking_prefix.mni

nonbreaking_prefix.mr

nonbreaking_prefix.nl

nonbreaking_prefix.or

nonbreaking_prefix.pa

nonbreaking_prefix.pl

basic-protected-patterns

deescape-special-chars-PTB.perl

deescape-special-chars.perl

delete-long-words.perl

detokenizer.perl

escape-special-chars.perl

lowercase.perl

normalize-punctuation.perl

pre-tok-clean.perl

pre-tokenizer.perl

pre_tokenize_cleaning.py

remove-non-printing-char.perl

replace-unicode-punctuation.perl

REGOLE COMUNI

VANTAGGI

- Adattarsi alle specifiche lingue
- Capacità di standardizzare il formato del testo tokenizzato
- Gestione della punteggiatura

SVANTAGGI

- Familiarità con il linguaggio di programmazione Perl
- Progettato per l'addestramento di modelli di traduzione automatica

Implementazione in Python

```
class MosesTokenizationPipeline:  
    def __init__(self, language='en'):  
        self.language = language  
        self.tokenizer = MosesTokenizer(self.language)  
  
    def tokenize_text(self, text):  
        # Tokenizza il testo e unisce i token in una stringa  
        return ' '.join(self.tokenizer(text))  
  
    def tokenize_column(self, df, column_name):  
        # Applica la tokenizzazione a una colonna del DataFrame  
        return df[column_name].apply(lambda x: self.tokenize_text(x) if isinstance(x, str) else x)  
  
    def close(self):  
        # Rilascia le risorse del tokenizzatore  
        self.tokenizer.close()
```

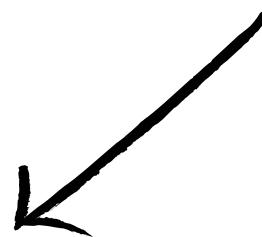
Pipeline di
Applicazione
della
tokenizzazione
con Moses
Tokenizer



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

Unigram LM

Un modello di linguaggio che si basa sulla probabilità di singole parole (unigrammi) senza considerare esplicitamente il contesto fornito dalle parole adiacenti. In questo modello, la probabilità di una parola viene determinata sulla base della frequenza con cui compare in un grande corpus di testo.



Creazione del vocabolario

Il tokenizzatore Unigram crea un vocabolario iniziale dal corpus di testo, identificando segmenti come parole e n-grammi di caratteri tramite degli algoritmi. Ogni segmento riceve una probabilità basata sulla sua frequenza nel testo, fondamentale per la tokenizzazione e l'ottimizzazione del vocabolario.

Ottimizzazione del vocabolario

Nell'ottimizzazione del tokenizzatore Unigram, si minimizza una funzione di perdita per valutare l'efficacia del vocabolario nel rappresentare il corpus. Attraverso la rimozione in batch, si riduce la dimensione del vocabolario mantenendo le caratteristiche linguistiche chiave per una rappresentazione accurata del testo.

Tokenizzazione del testo

Durante la tokenizzazione del testo con il tokenizzatore Unigram, si usa l'algoritmo di Viterbi per suddividere il testo in token, post ottimizzazione del vocabolario. L'algoritmo identifica e valuta tutte le possibili sequenze di sottoparole, scegliendo quella con la più alta probabilità di rappresentare fedelmente il testo.

VANTAGGI

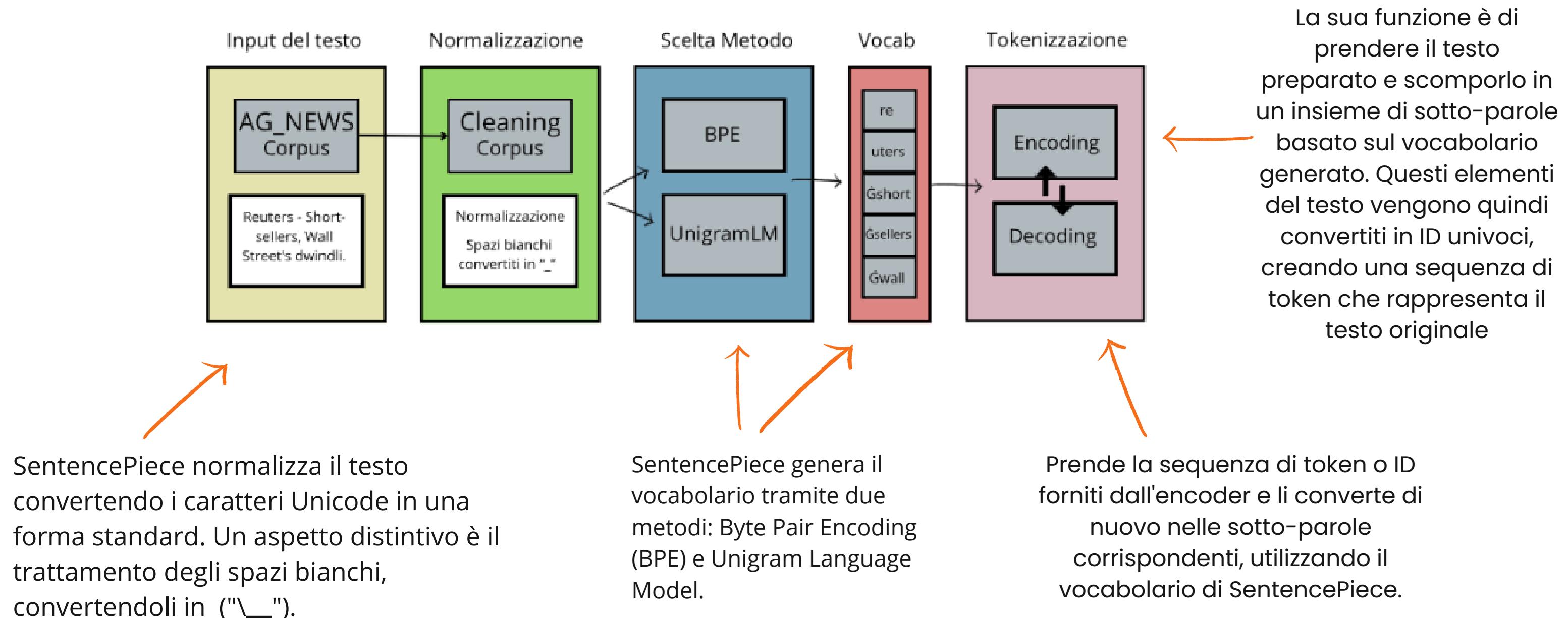
- Adattabilità linguistica efficace per lingue con strutture morfologiche complesse e senza spazi bianchi
- Gestione efficiente delle parole fuori dal vocabolario (OOV) attraverso la scomposizione in sotto-unità note.
- Flessibilità nella tokenizzazione basata sulle frequenze delle sotto-parole nel corpus.
- Utilità in contesti multilingue

SVANTAGGI

- 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.

Sentencepiece

SentencePiece è uno strumento di tokenizzazione e detokenizzazione di sotto-parole, introdotto da Google nel 2018, che lavora direttamente su frasi grezze senza pre-elaborazione, rendendolo ideale per lingue senza chiara segmentazione delle parole.



VANTAGGI

- Indipendenza dalla Lingua
- Gestione delle Parole Fuori dal Vocabolario (OOV)
- Flessibilità nel Tokenizzare
- Efficienza in Contesti Multilingue
- Miglioramento della Coerenza e della Precisione

SVANTAGGI

- Complessità nel Gestire Vocabolari Grandi
- Mancanza di Sensibilità al Contesto
- Overfitting con Testi Specifici
- Prestazioni Varie con Diverse Lingue
- Risorse Computazionali per l'Addestramento

Implementazione in Python

XLNet è un modello di lingua preaddestrato che utilizza una variante di autoregressione condizionata ed è stato addestrato su un corpus che include BooksCorpus, English Wikipedia, Giga5, ClueWeb 2012-B e Common Crawl e utilizza una tokenizzazione sentencepiece.

```
def transform(self, X, **transform_params):
    # Applica la tokenizzazione XLNet a ogni elemento
    return [tokenizer.encode(text, add_special_tokens=True) for text in X]

def fit(self, X, y=None, **fit_params):
    return self
```

Processo di
Tokenizzazione

```
# Creazione della pipeline
pipeline = Pipeline([
    ('xlnet_tokenization', XLNetTokenization())])

# Esempio di utilizzo
train_set = pd.read_csv('dataset/train_set_bpe.csv')
test_set = pd.read_csv("dataset/test_set_bpe.csv")

train_set['XLN'] = pipeline.fit_transform(train_set['cleaned_text'])
test_set['XLN'] = pipeline.fit_transform(test_set['cleaned_text'])

#rimozione liste
train_set['XLN'] = train_set['XLN'].apply(lambda x: ' '.join(map(str, x)) if isinstance(x, list) else x)
test_set['XLN'] = test_set['XLN'].apply(lambda x: ' '.join(map(str, x)) if isinstance(x, list) else x)
```

←
Pipeline di
Applicazione della
Tokenizzazione



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

DATASET AG_NEWS

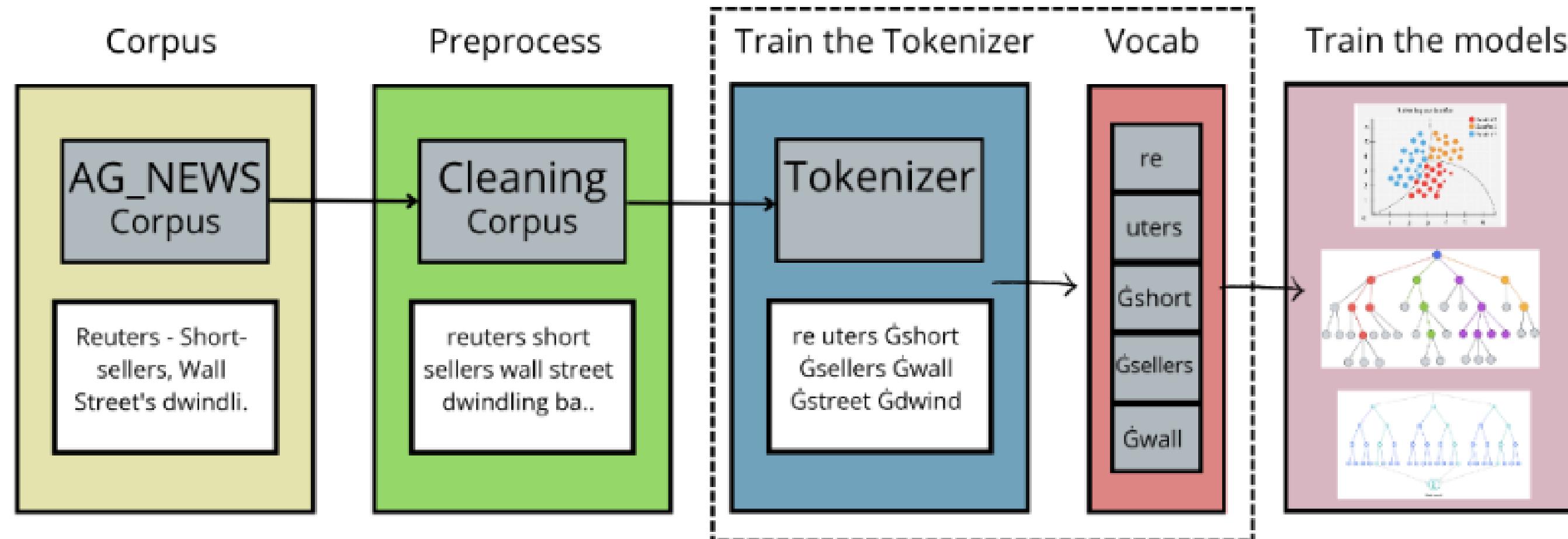
This file consists of 7600 testing samples of news articles that contain **3 columns**. The first column is Class Id, the second column is Title and the third column is Description. The class ids are numbered 1-4 where 1 represents World, 2 represents Sports, 3 represents Business and 4 represents Sci/Tech.

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

# Class Index	=	A Title	=	A Description	=
Consists of class ids 1-4 where 1-World, 2-Sports, 3-Business, 4-Sci/Tech		Contains title of the news articles		Contains description of the news articles	
					
1		7568		7594	
		unique values		unique values	

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 acquis

MODUS OPERANDI



ANALISI DEI RISULTATI

**Tempo di Addestramento
e Velocità di
Tokenizzazione**

**Analisi Comparativa
in termini di
Memoria**

**Analisi Comparativa in
termini di Output di
Tokenizzazione**

**Analisi Comparativa
in relazione
all'accuracy nei
Modelli di ML**



Tempo di Addestramento e Velocità di Tokenizzazione

Metriche

01

DISPOSITIVO: Macbook con
Chip M1

02

DATASET:

- 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
- Titoli di addestramento dal peso di 5,2 MB

Risultati

(per task di classificazione delle notizie)

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

Necessità di Rapido Addestramento e Applicazione in Contesti di Dati Limitati

White Spaces & Moses

Non richiedendo un processo di addestramento specifico, sono rapidi da implementare e applicare. Uno svantaggio è la mancanza di flessibilità e potenziale limitatezza nell'analisi di testi complessi,

Contesti di Ampie Basi di Dati con Necessità di Prestazioni Elevate

RobertaBPE, SP con XLNet

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. Un limite è che il loro corpus di riferimento potrebbe non adattarsi perfettamente ai requisiti specifici di un nuovo task.

Necessità di Personalizzazione e ampie prestazioni computazionali

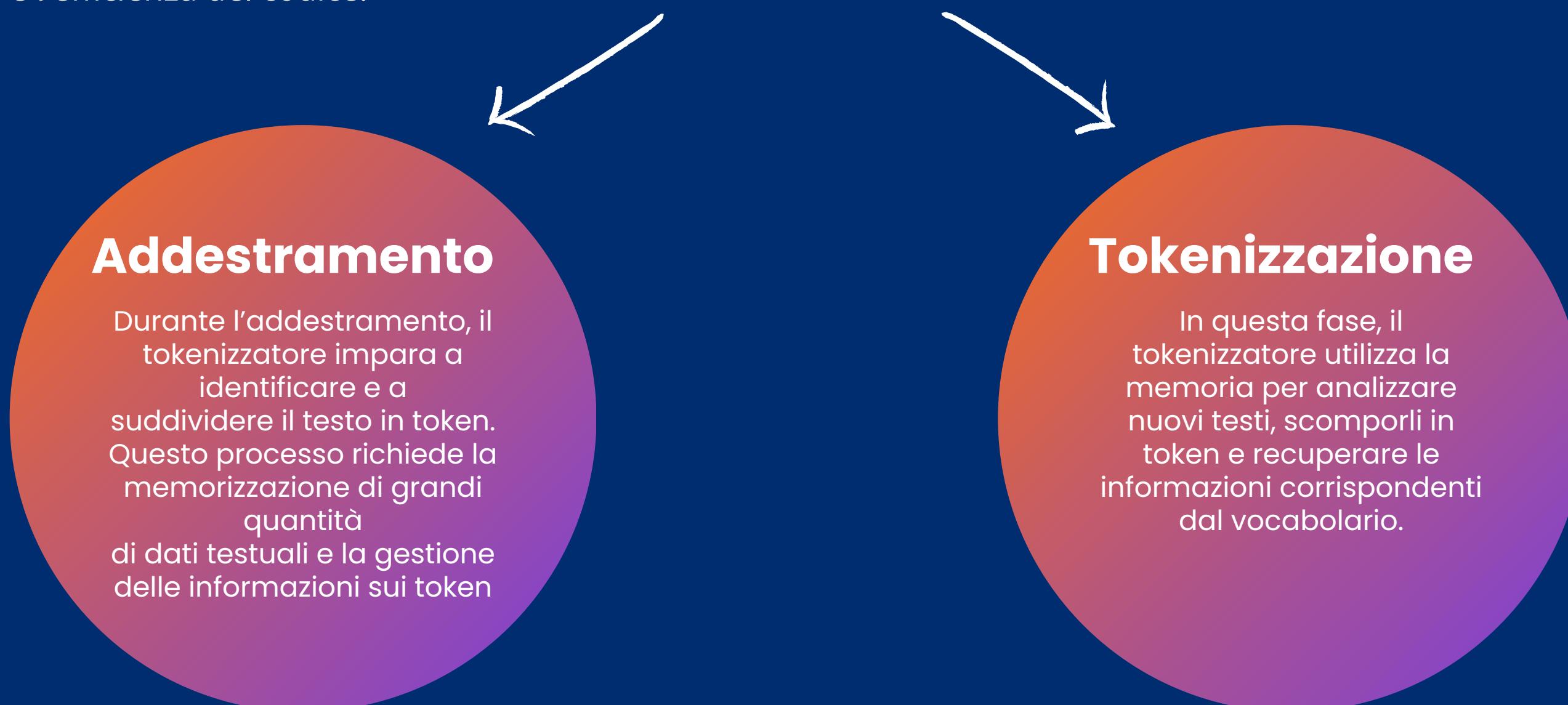
BPE & Morfessor

Capacità di essere addestrati su corpora specifici, offrendo una personalizzazione elevata, ma richiedono tempi di addestramento più lunghi

Analisi Comparativa in termini di Memoria

Metriche

MEMORIA: La memoria, 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.



Risultati

(per task di classificazione delle notizie)

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

**Task con Risorse Computazionali
Limitate e semplicità di
applicazione**

White Spaces & Moses

Non richiedendo quasi alcuna memoria per l'addestramento perché si limitano a dividere il testo in token basandosi sugli spazi/regole. Si nota che la lingua di contesto è l'inglese altrimenti tali risultati non sarebbero replicabili per scenari più complesse non delimitate da spazi come Cinese per WS, mentre per Moses non vi è tale problema.

**Task con Risorse Computazionali
Limitate ma necessità di
implementazione ottimizzate**

RobertaBPE, SP con XLNet

Scelta appropriata per situazioni in cui le risorse computazionali durante la fase di addestramento sono limitate, ma si desidera comunque beneficiare di un vocabolario ottimizzato. Essendo già pre-addestrati, non richiedono risorse aggiuntive durante l'addestramento e offrono un processo di tokenizzazione efficiente per una varietà di contesti linguistici.

**Task con Risorse Computazionali
Elevate e necessità di
personalizzazione**

BPE & Morfessor

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. (BPE)

Analisi Comparativa in termini di Output di Tokenizzazione

Quando un nuovo testo viene processato, il tokenizzatore lo scomponete in una serie di token basandosi sul vocabolario esistente. Ogni parola, frase o simbolo nel testo viene confrontato con il vocabolario e sostituito con l'identificativo corrispondente del token.



L'output della tokenizzazione, quindi, è una sequenza di quesiti identificativi che rappresenta fedelmente il testo originale, ma in un formato che è più facilmente elaborabile dai successivi passaggi del pre-processing

Risultati

(per task di classificazione delle notizie)

Tokenizzatori	Testo	Nº Token
White Spaces	unions representing workers turner newall say disappointed talks stricken parent firm federal mogul	13
BPE	un ions Ģre p res ent ing Ģw ork ers Ģt urn er Ģnew all Ģs ay Ģd is app o int ed Ģtalks Ģst ric k en Ģp ar ent Ģf irm Ģf ed er al Ģm og ul	40
BPE Roberta	un ions Ģrepresenting Ģworkers Ģturn er Ģnew all Ģsay Ģdisappointed Ģtalks Ģstricken Ģparent Ģfirm Ģfederal Ģmogul'	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

Testi con Costrutti Semplici, Velocità e Output di tokenizzazione Ridotto

White Spaces & Moses

Nel panorama della lingua inglese, i tokenizzatori che si dimostrano più efficaci per questi compiti sono quelli che riescono a gestire strutture linguistiche semplici, fornendo al contempo un numero limitato di token.

Utilizzo dell'Essenza di un Tokenizzatore e Output Non Troppo Numeroso

RobertaBPE, SP con XLNet

I tokenizzatori che sono stati pre-addestrati offrono la capacità di elaborare strutture linguistiche complesse, beneficiando del loro solido fondamento teorico. Essi sono in grado di catturare efficacemente l'essenza del tokenizzatore su cui sono basati, essendo stati ottimizzati da team di specialisti in modo più professionale rispetto all'impiego personalizzato di tokenizzatori di base.

Task Specializzati che richiedono personalizzazione ma anche capacità computazionali e di dominio

BPE Personalizzato

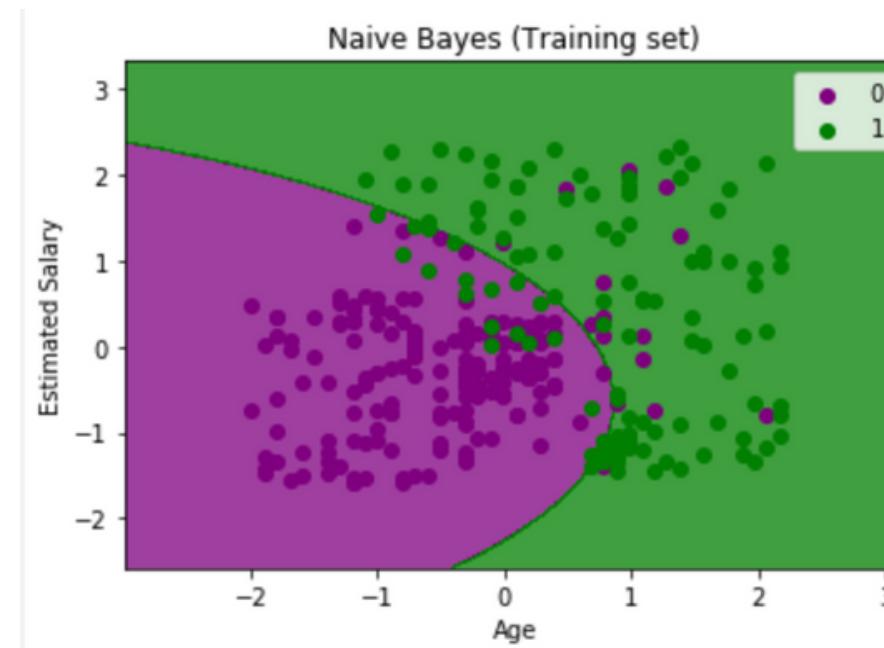
Qualora un compito richieda una specializzazione più specifica, l'impiego di un BPE personalizzato si rivela la scelta migliore, a condizione che si disponga delle risorse computazionali necessarie per la fase di addestramento del tokenizzatore e delle competenze appropriate per affinare i parametri specifici. Ciò evita di ottenere risultati inadeguati.
.

Analisi Comparativa relativa all'accuracy nei Modelli di ML

Metriche

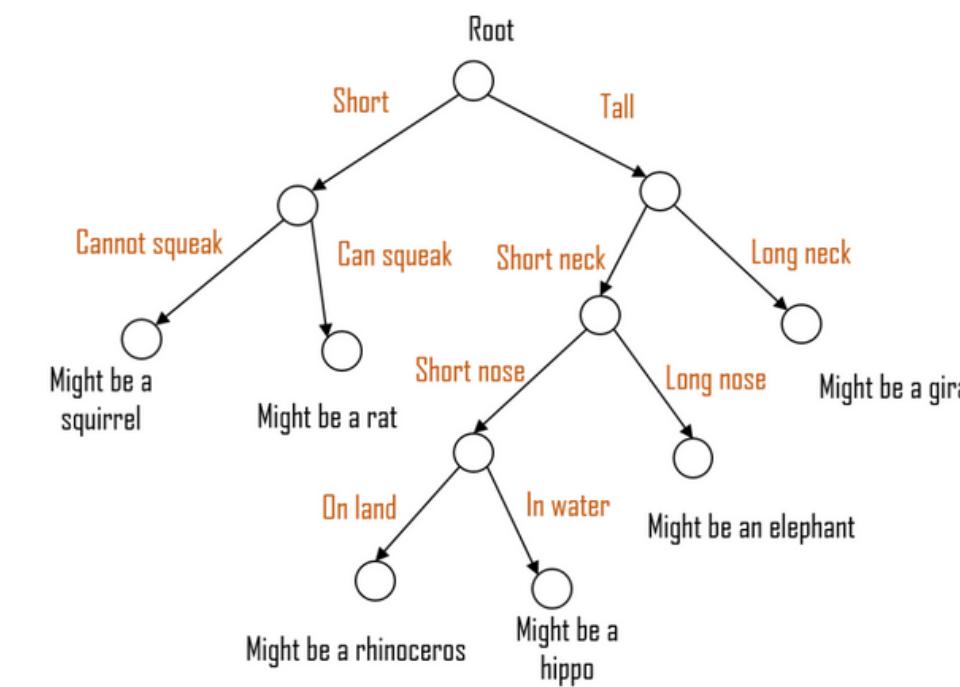
01

NAIVE BAYES



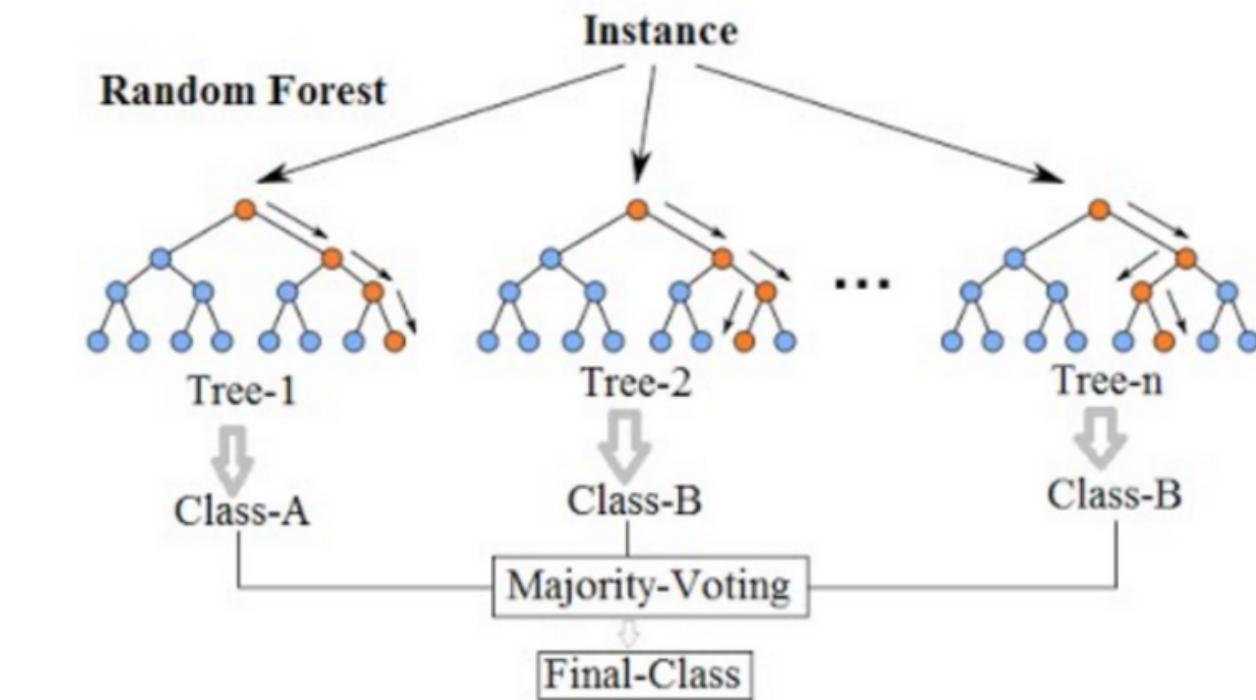
02

DECISION TREE



03

RANDOM FOREST



Risultati

(per task di classificazione delle notizie)

	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

Naive Bayes

Naive Bayes è un modello probabilistico, che dipende dall'ipotesi che le caratteristiche (in questo caso, i token del testo) siano indipendenti. Tokenizzatori semplici, come quello basato su spazi bianchi, si allineano bene con l'assunzione di indipendenza di Naive Bayes, mentre tokenizzatori più avanzati, come BPE Roberta e XLNet, mostrano che la qualità dei token può essere fondamentale per migliorare la performance del modello.

Decision Tree

Si è riscontrato che i tokenizzatori Moses e White space migliorano l'accuracy, suggerendo che una tokenizzazione più semplice favorisce le divisioni ottimali del Decision Tree, mentre il BPE Personalizzato ha mostrato prestazioni inferiori, probabilmente a causa di un output di tokenizzazione troppo ampio e meno efficace.

Random Forest

La superiorità di Random Forest con alcuni tokenizzatori si deve alla sua natura di modello ensemble e alla tipologia di dati. Il tokenizzatore White space, semplice ed efficace, e Moses, ottimizzato per frasi standardizzate, offrono feature utili alla classificazione. I complessi BPE Roberta e XLN forniscono set di dati dettagliati e contestuali, permettendo a Random Forest di cogliere pattern complessi e migliorare la classificazione.

BIBLIOGRAFIA

- Hugging Face : "Byte-Pair Encoding Tokenization." Disponibile su: (<https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt>).
 - RoBERTa.nformazioni dettagliate disponibili su: (https://huggingface.co/docs/transformers/model_doc/roberta).
 - Unigram Tokenization: Approfondisci su: (<https://huggingface.co/learn/nlp-course/chapter6/7?fw=pt>).
- Philipp Koehn: Moses: Statistical Machine Translation System, User Manual and Code Guide.
" Università di Edimburgo, 2022.
- Philipp Koehn et al."Moses: Open Source Toolkit for Statistical Machine Translation.
" Associazione per la Linguistica Computazionale, 2007.
- Taku Kudo."Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates." Google, Inc., 2018.
 - Taku Kudo e John Richardson.
"SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing." Google, Inc., 2018.
- Peter Smit. "Morfessor 2.0: Toolkit for Statistical Morphological Segmentation." Dipartimento di Scienze dell'Informazione e Informatica, Università di Aalto, 2014.
- CAGRI TORAMAN et al. "Impact of Tokenization on Language Models: An Analysis for Turkish." Aselsan Research Center, Turchia, 2022.
- Sami Virpioja, Peter Smit e Miko Kurimo.
"Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline." Dipartimento di Scienze dell'Informazione e Informatica, Università di Aalto, 2013
- Yang, Zhilin et al. «XLNet: Generalized Autoregressive Pretraining for Language Understanding». In: CoRR (2019). URL: <http://arxiv.org/abs/1906.08237>.
- Zouhar, Víťem et al. «A Formal Perspective on Byte-Pair Encoding». In: ETH Zürich (2023).
- Sabrina J. Mielke et al."Between Words and Characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP." BigScience Workshop Tokenization Working Group, 2021.
- Awaldeep Singh."Hugging Face: Understanding Tokenizers." Medium, 2022.