Scadenze:
Presentazione domanda e titolo: entro il 22 Novembre
Consegna riassunto: dal 22 novembre al 6 dicembre
Consegna tesi: entro il 6 dicembre
Seduta laurea: dal 13 al 17 dicembre (ancora da definire)

UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Informatica

TITOLO DA DEFINIRE

Relatore: Prof. Alfio FERRARA

Correlatore: Dr. Francesco PERITI

Autore: Sebastiano Caccaro
Matricola: 958683

Anno Accademico 2020-2021

dedicato a ...

Prefazione

hkjafgyruet.

Organizzazione della tesi

La tesi è organizzata come segue:

- nel Capitolo 1 ....

Ringraziamenti

asdjhgftry.

Indice

Capitolo 1

Introduzione

Capitolo 2

Stato dell'Arte

In questo capitolo bla bla bla sezione .... Aggiungere

2.1 Introduzione

Optical Character Recognition Ad oggi, sempre più libri cartacei, riviste e giornali presenti in biblioteche e archivi storici stanno venendo trasformati in versioni elettroniche che possono essere manipolate da un computer. A questo scopo, nel corso degli anni sono state sviluppate tecnologie di Optical Character Recognition (comunemente abbreviato con OCR) per tradurre le scansioni e immagini di documenti testuali in testo interpretabile e processabile da un computer. Questi sistemi, però, non sono perfetti e possono introdurre errori nel testo. Può accadere, infatti, che durante il processo di scansione alcuni caratteri vengano letti in modo errato, altri vengano aggiunti e altri ancora non riconosciuti. È, ad esempio, particolarmente probabile che caratteri o sequenze di caratteri graficamente simili come *"li"* e *"n"* vengano scambiati fra di loro[1]. La frequenza di tali errori è influenzata da fattori quali la condizione di deterioramento di un documento e la qualità di acquisizione dell'immagine[2]: la presenza di granelli di polvere, caratteri scoloriti, pagine ingiallite o artefatti risultati dalla scansione, ad esempio, influiscono negativamente sulle performance dei sistemi OCR.
La presenza di tali errori in corpora acquisiti tramite OCR risulta problematica in quanto rende meno precisi task di Natural Language Processing (NLP) come, ad esempio, l'esecuzione di query [3] o il topic modelling[4]. Per ovviare a tali problemi, sono state sviluppate varie soluzioni che mirano a minimizzare il quantitativo di errori presenti nel testo estratto. È possibile classificare queste soluzioni nelle seguenti due categorie:

- **OCR Pre-processing**: ricadono in questa categoria tutte quelle tecniche che mirano ad ottenere migliori risultati dall'estrazione del testo attraverso il

miglioramento dell'input, ovvero delle immagini, che viene usato dai software di OCR. Tali metodi includono, ma non si limitano a, l'uso di migliori tecniche di scansione, la correzione del contrasto nell'immagine[5] e la rotazione e correzione di deformazioni nell'immagine[6] (Figura 1).

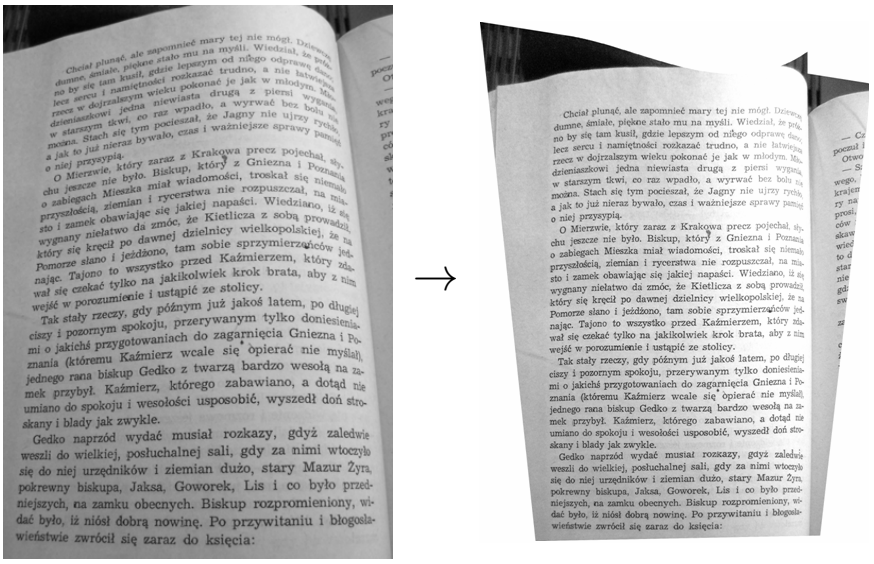

Figura 1: A sinistra foto di una pagina contenente del testo. A destra, foto della stessa pagina pre-processata per facilitare l'estrazione del testo. Esempio preso da [6].

- **OCR Post-processing**: ricadono in questa categoria tutte quelle tecniche che mirano ad individuare e correggere gli errori presenti nell'output generato dai vari software di OCR. Essendo l'OCR Post-processing oggetto di questa tesi, sarà approfondito a parte nella sezione 2.2.

OCR pre-processing e post-processing sono spesso usati in congiunzione per ottenere migliori risultati dall'estrazione del testo.

2.2 OCR Post-processing

In letteratura sono presenti numerosi approcci al problema dell'OCR post-processing, molti dei quali adottano strategie molto differenti. Dato ciò, non è possibile delineare una metodologia generale che ogni approccio segue, ma, in generale, ogni approccio deve:

1. **Identificare gli errori** (Error Detection), ovvero delimitare tutte le sezioni contenenti errori nel testo, senza delimitare sezioni corrette. Espandere?

2. **Correggere gli errori** (Error Correction), ovvero ripristinare testo originale nelle sezioni individuate in precedenza. Espandere?

Nelle seguenti sottosezioni sono quindi esposti alcuni dei principali approcci per letteratura, raggruppati nelle seguenti categorie di metodologie:

- Approcci basati su n-grams
- Approcci basati su NMT
- Approcci basati su BERT
- Altri approcci

Controllo che non sia cambiato niente

2.2.1 Approcci basati su n-grams

Per discutere gli approcci basati sugli n-grams è prima necessario definire i concetti di token, tokenizzazione e n-gram.

Token È riportata la definizione fornita in [7]: "Un token è una stringa di caratteri contigui compresi fra due spazi, o fra uno spazio e un segno di punteggiatura. Sono token anche [numeri] interi, [numeri] reali o numeri contenenti i due punti (ore, ad esempio 2:00). Tutti gli altri simboli sono considerati essi stessi dei token, eccetto gli apostrofi e i punti di domanda attaccati ad una parola (senza spazi), che in molti casi rappresentano acronimi o citazioni."
Più informalmente è possibile associare il concetto di token a quello di parola nel linguaggio naturale.

Tokenizzazione Data la precedente definizione di token, per tokenizzazione si intende il dividere un testo, una frase, o più in generale una stringa nei token che la compongono. Data quindi una frase $f \in F$, tokenizzare una frase vuol dire applicare una funzione:

$$Tok : F \to T \tag{1}$$

dove ogni $t \in T$ è una lista $[t_1, ..., t_n]$ in cui ogni t_i è un token appartenente alla frase iniziale. Più informalmente quindi, la tokenizzazione restituisce le singole parole appartenenti alla frase iniziale. Ad esempio, data la frase f_{es}:

"Cantami, o Diva, del pelide Achille l'ira funesta che infiniti addusse lutti agli Achei"

la sua versione tokenizzata $Tok(f_{es})$ è:

["Cantami", ",", "o", "Diva", ",", "del", "pelide", "Achille", "l'", "ira", "funesta", "che", "infiniti", "addusse", "lutti", "agli", "Achei"]

n-gram Un n-gram è una sottosequenza contigua di n elementi di una data sequenza [8]. Gli elementi in questione possono essere fonemi, sillabe, lettere parole ecc. Nel proseguo di questo documento ogni riferimento a n-gram, salvo indicazione contraria, si riferisce a n-gram di token. Gli n-gram trovano ampio uso nel campo del NLP, dove sono usati, ad esempio, per creare modelli linguistici statistici.
In seguito è mostrato un esempio di scomposizione di una frase in n-gram di lunghezza 3, detti quindi 3-gram o trigrams. Dato la frase del precedente esempio f_{es}, e data la sua scomposizione in token $Tok(f_{es})$, i trigrams formati sono i seguenti:

["Cantami", ",", "o"], [",", "o", "Diva"], ["o", "Diva", ","], ["Diva", ",", "del"], [",", "del", "pelide"], ["del", "pelide", "Achille"], ["pelide", "Achille", "l'"], ["Achille", "l'", "ira"], ["l'", "ira", "funesta"], ["ira", "funesta", "che"], ["funesta", "che", "infiniti"], ["che", "infiniti", "addusse"], ["infiniti", "addusse", "lutti"], ["addusse", "lutti", "agli"], ["lutti", "agli", "Achei"]

Approcci basati su n-grams Gli approcci descritti in questa sezione utilizzano modelli linguistici basati su n-gram per individuare e correggere gli errori. Le soluzioni proposte in questa sezione fanno entrambe uso del Google Web 1T 5-gram dataset[9], che da qui in poi verrà riferito come GW5. GW5 è un dataset contenente n-grams in lingua inglese (da unigrams a 5-grams) associati alla loro frequenza osservata su un totale di 1 trilione di parole. Tutti gli n-grams sono stati estratti attraverso il crawling di pagine web. L'enorme scala del database e la metodologia tramite la quale è stato ottenuto comporta che sia possibile estrarre da GW5 un amplio lessico che può essere affabilmente usato per fare error detection. Per lo stesso motivo, il dataset si presta bene anche all'applicazione in campi con terminologie di nicchia o altamente specifiche.
Il primo approccio trattato quello presentato in [10]. L'approccio è diviso di tre fasi:

1. Error Detection: sono usati gli unigram in GW5 per identificare gli errori. Ogni token all'interno del testo da correggere non presente nella lista degli unigram è considerato un errore. È quindi chiaro come questo metodo riesca ad individuare (e quindi correggere) solo i non-word errors, ovvero tutti quelli errori che risultano in parole non presenti in un dato lessico. Non sono trattati da questo approccio i real-word errors, ovvero tutti quelli errori che risultano in una parola presente in un dato lessico. Si pensi ad esempio alla parola *"sale"* interpretata come *"sala"* da un software OCR.

2. Candidate Spelling Generation: per ogni errore si produce una lista di parole candidate per la correzione. Per fare ciò si scompone la parola errata in 2-grams a livello di carattere. Ad esempio, la parola "sangle" è scomposta in "sa", "an", "ng", "gl", "le". Per ognuna delle parole nel lessico di unigram, in seguito, si conta quante occorrenze dei 2-gram della parola da correggere sono contenute. Ad esempio, la parola "single" contiene tre occorrenze ("ng","gl","le"). Le prime 10 parole con più occorrenze dei 2-gram sono considerate i candidati per la correzione.

3. Error Correction: si considera il 5-gram terminante con la parola errata $[t_1, t_2, t_3, t_4, err]$. Per ognuno dei candidati c_i è prodotto il 5-gram $[t_1, t_2, t_3, t_4, c_i]$: di questi 5-gram prodotti, quello con più occorrenze all'interno di GW5 contiene la correzione da applicare.

Un approccio simile è esposto in [11]. A differenza dell'approccio precedente, l'error detection e la generazione dei candidati non usano GW5, ma sono usate altre tecniche che mirano a correggere anche i real-word errors. L'approccio utilizzato per l'error correction invece sfrutta lo stesso principio di [10], ma con una logica leggermente più complessa. Il funzionamento è il seguente:

1. Error Detection e Candidate Generation: per i non-word errors GNU-Aspell[12] è utilizzato per individuare gli errori e proporre i possibili candidati per la correzione. Per i real-word errors invece, sono usati dei confusion-set pre-definiti per individuare i possibili errori e generare i candidati. Un confusion-set non è altro che un insieme di parole simili che possono essere confuse fra di loro, come {they' re, their, there}. I candidati per un possibile errore sono quindi le parole appartenenti al suo confusion set.

2. Error Correction. Per ogni candidato si considera un intorno di 2 parole, andando così a comporre un 5-gram. Se il 5-gram in cui è presente l'errore nel testo è $[t_1, t_2, err, t_3, t_4]$, allora per il candidato c_i sarà $[t_1, t_2, c_i, t_3, t_4]$. Il candidato scelto è quello il cui 5-gram compare più volte in GW5. In caso nessun 5-gram compaia nel dataset, il processo si ripete con il 3-gram $[t_2, c_i, t_3]$ e successivamente solo con l'unigram $[c_i]$, ovvero viene scelto il candidato con la maggior frequenza nel corpus.

Gli approcci fin'ora descritti, seppur molto efficaci contro i non-word errors, ma non correggono o sono poco efficaci contro i real-word errors e altri tipi di errori. Si pensi ad esempio a situazioni in cui token viene separato da spazi, o in cui due token sono fusi insieme. Inoltre questi approcci, richiedono un'elevata quantità di dati per funzionare efficacemente (GW5 occupa 87GiB su disco) , il che non li rende facilmente applicabili.

2.2.2 Approcci basati su NMT

2.2.3 Approcci basati su BERT

2.2.4 Altri Approcci

Capitolo 3

Dataset e Perturbazione

Capitolo 4

Metodologia di correzione

In questo capitolo è descritta la metodologia di correzione.
Nella sezione 4.1 vengono descritti gli obiettivi del processo di correzione e le criticità che lo contraddistinguono. Nella sottosezione 4.2.1 è presente una panoramica generale del processo di correzione. Sono inoltre descritte le fasi e le componenti del sistema. Nella sottosezione 4.2.2 è descritto il funzionamento del BERT Masked Language Model. Nelle sottosezioni 4.2.3 e 4.2.4 è descritto il funzionamento dei moduli che stanno alla base del sistema di correzione.

4.1 Introduzione

L'Optical Character Recognition, o OCR, è una tecnologia tramite la quale è possibile estrarre e digitalizzare il testo presente in un'immagine. Più precisamente, "digitalizzare" significa tradurre il testo dell'immagine in una una codifica leggibile da una macchina, come ad esempio ASCII o Unicode.

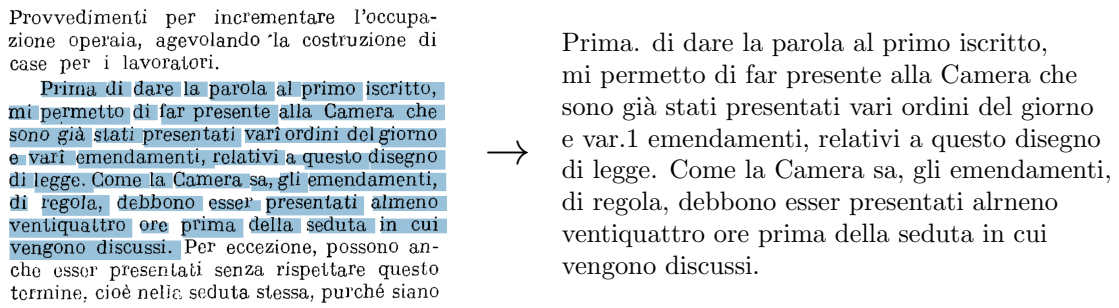

Figura 2: A sinistra un frammento di immagine contenente testo, a destra il testo digitalizzato estratto dal frammento di immagine.

L'uso di sistemi OCR è particolarmente vantaggioso per l'archiviazione dei documenti. Infatti, l'estrazione del testo rende possibile eseguire sui documenti operazioni di ricerca e di reperimento informazioni. Si pensi a un'operazione basilare come la ricerca di tutti i documenti che contengono una determinata parola: un task dispendioso e manuale che è quasi istantaneo se si ha a disposizione il testo digitalizzato.

Errori I sistemi OCR presentano però alcuni problemi che possono influenzare negativamente le performance nel reperimento di informazioni [3][4]. Si veda, ad esempio, il testo estratto in Figura 2: seppur la maggior parte delle parole sono state riconosciute correttamente, è possibile notare alcuni errori. Generalmente la presenza di tali errori è dovuta a piccole imprecisioni nell'immagine iniziale: si pensi ad esempio un granello di polvere che può essere scambiato per un punto, o ad una *"n"* battuta male che viene scambiata per la sequenza *"ii"*. Si possono distinguere le seguenti categorie di errore:

- **Word Error**. Aggiunta, rimozione o sostituzione di caratteri spuri all'interno di una parola. Questo tipo di errore si divide in due ulteriori sottocategorie:
  - Non-word error (NW): la parola affetta da errore non è presente in un dato vocabolario. Un errore di questo tipo è la parola *"gioia"* che diventa *"g1o1a"*.
  - Real-word error (RW): la parola affetta da errore è presente in un dato vocabolario, ma non è corretta nella frase in cui è inserita. Ad esempio, la frase *"qualvolta i popoli ripongono la loro fiducia nelle armi e nella guerra"* può essere erroneamente interpretata come *"qualvolta i popoli ripongono la loro fiducia nelle ami e nella guerra"*. "ami" è una parola presente nel vocabolario, ma non è corretta nel contesto della frase data.
- **Word Segmentation Error**. Si ha un errore di segmentazione quando aggiunte, rimozioni o sostituzioni di caratteri (incluso il carattere spazio) occorrono in maniera tale da dividere o unire parole. Sono considerati errori di segmentazione anche aggiunte di caratteri e segni di punteggiatura spuri all'interno del testo. Sono distinte le seguenti sottocategorie di errori di segmentazione:
  - Space Splitting (SP), quando i caratteri di una parola o più parole adiacenti sono intervallati da spaziature. Ad esempio, si ha un errore di questo tipo nella frase *"vostra presenza conferma l'attaccamento alla cattedra di Pietro e la fedeltà al suo Magistero"* che viene letta come

"vostra presenza conferma l'a t t a c c a m e n t o a l l a cattedra di Pietro e la fedeltà al suo Magistero".

- Punctuation Splitting (DP), quando uno o più segni di punteggiatura dividono in più parti una parola. Ad esempio, data la parola *"attaccamento"* si ha un errore DP quando essa viene interpretata come *"at,tacc,amento"*.
- Punctuation Insertion (PS), quando uno o più segni di punteggiatura vengono aggiunti fra una parola e l'altra. Ad esempio, data la frase *"Per secoli la Chiesa ha patrocinato artisti che hanno..."*, si hanno errori PS quando essa viene interpretata come *"Per secoli'la Chiesa ha.,patrocinato artisti che hanno..."*.

In Tabella 1 sono riportati ulteriori esempi degli errori appena elencati.

Originale	**OCR**	**Tipo**	**Errore**
dell'impresa	dell'iinpresa	NW	Sostituzione di "m" con "in"
interessano	iateressano	NW	Sostituzione di "n" con "a"
questo modo	questo nodo	RW	Sostituzione di "m" con "n"
l'ingordizia	I'ingordizia	RW	Sostituzione di "l" con "I"
produttrice	pro d u t t rice	SP	Parola spezzettata da spazi
azionisti	azi:misti	DP	Sostituzione con divisione della parola
vicinanze del	vicinanze .del	PS	Introduzione di punteggiatura

Tabella 1: Esempi di errori

Se, come già detto, gli errori derivano spesso da piccoli difetti nell'immagine di partenza, è lecito aspettarsi una maggior quantità di errori da documenti più deteriorati o datati, come archivi storici. Per lo stesso motivo l'intensità degli errori può variare, a seconda delle condizioni in parti diverse di uno documento.

OCR Post-processing Una strategia spesso adottata per minimizzare il numero di errori è l'aggiunta di una fase di post elaborazione (OCR post-processing) in seguito all'acquisizione dei documenti tramite OCR. Lo scopo della fase di OCR post-processing è quello di correggere gli errori introdotti durante l'estrazione del testo dalle immagini. Una delle difficoltà in questa fase sta nel non introdurre nuovi errori nel testo. Si pensi ai seguenti casi:

- Il sistema prova a correggere una parola che non contiene errori. In questo caso, qualsiasi correzione risulterà in un nuovo errore. Una delle maggiori criticità nello sviluppare il sistema di correzione, infatti, sta nell'identificare correttamente gli errori all'interno del testo.

- Il sistema identifica correttamente un errore, ma propone una correzione sbagliata. Si pensi al seguente esempio: *"vesodvi"* viene erroneamente corretto in *"vedovi"*. Tuttavia, la parola originale era *"vescovi"*.

La metodologia di correzione sviluppata è modellata come una pipeline che si compone di una serie di moduli ripetibili in sequenza. Ogni modulo è una funzione che si occupa di individuare e correggere una specifica categoria di errore.
La metodologia sviluppata si propone di correggere gli errori di tipo RW e SP.

4.2 Architettura del sistema

4.2.1 Panoramica Generale

Il sistema di correzione è articolato come una pipeline composta da uno o più moduli concatenati. Un modulo è una funzione il cui scopo è correggere una specifica tipologia di errori all'interno di una frase. Un modulo può essere formalmente definito come segue:

$$M : F \mapsto F\prime \tag{2}$$

dove F e $F\prime$ sono rispettivamente la frase originale e quella con delle correzioni apportate.
Sono stati definiti i seguenti moduli:

- **Modulo di correzione Token** (M_{TK}): individua e corregge tutti gli errori di tipo RW. È discusso in dettaglio nella sottosezione 4.2.3.
- **Modulo di correzione Split** (M_{SP}): individua e corregge tutti gli errori di tipo SP. È discusso in dettaglio nella sottosezione 4.2.4.

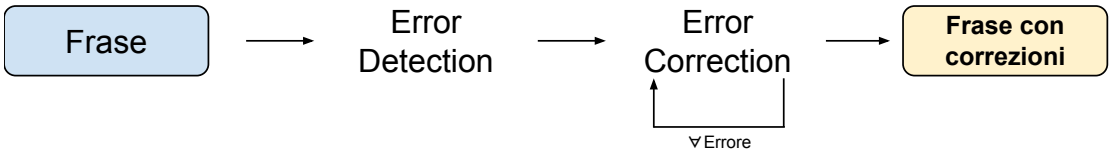

Figura 3: Schema del funzionamento di un modulo

Come si può vedere in Figura 3, il funzionamento di un modulo si compone delle fasi Error Detection (che da qui in poi sarà abbreviata con ED) ed Error Correction (che da qui in poi sarà abbreviata con EC). Durante la fase di ED vengono individuati gli errori presenti all'interno dalla frase da correggere: dato

che moduli differenti correggono tipologie di errore differenti, ogni modulo applica una diversa strategia per individuare gli errori. Nella fase di EC vengono effettuate le correzioni degli errori individuati nella fase precedente.

Esempio Si supponga di voler applicare alla seguente frase il modulo di correzione token:

"Q o a l c h n Papa nort è compreso in questa comunità d1 morti gluriosi"

Il modulo di correzione token riconosce e corregge solo gli errori di tipo NW (Non-word error, sottolineati con linea continua), mentre ignora tutti gli errori di altro tipo (sottolineati con linea tratteggiata). L'output del modulo, supponendo di riuscire a correggere tutti gli errori, è il seguente:

"Q o a l c h n Papa non è compreso in questa comunità di morti gloriosi"

Data la definizione di modulo, è possibile definire una pipeline di correzione come una concatenazione di uno o più moduli. Più formalmente, una pipeline di correzione è definita come segue:

$$P : f_1 \circ f_2 \circ ... \circ f_n \tag{3}$$

dove ogni f_i è un modulo.

Figura 4: Schema riassuntivo della pipeline di correzione

Esempio Continuando l'esempio precedente, si supponga di concatenare un modulo M_{SP} al precedente modulo M_{TK}. Il modulo riceve come input la frase con le correzioni apportate dal modulo precedente, e corregge tutti gli errori di tipo SP presenti nella frase (ovvero quelli con la sottolineatura tratteggiata):

"Qualche Papa non è compreso in questa comunità di morti gloriosi"

All'interno della pipeline di correzione possono essere presenti più occorrenze di uno stesso modulo. In frasi con molti errori una sola applicazione di un dato modulo di correzione potrebbe non essere sufficiente per correggere tutti gli errori

della propria tipologia. Per ragioni legate al funzionamento del BERT Masked Language Model che verranno meglio chiarite nelle prossime sottosezioni, infatti, ripetere un dato modulo più volte può aumentare tangibilmente le performance di correzione.

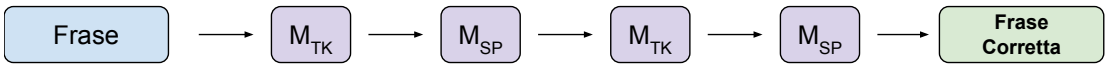

Figura 5: Esempio di una possibile pipeline con più occorrenze di uno stesso modulo

4.2.2 BERT Masked Language Modeling

La fase di error correction di entrambi i moduli implementati fa uso del BERT Masked Language Modeling (da qui in poi riferito come BERT MLM). Data una qualsiasi sequenza $S = [w_1, ..., w_n]$ nella quale ogni w_i è una parola, è possibile mascherare una sola parola w_i con la stringa *[MASK]*. Si ottiene così la sequenza con maschera $S\prime = [w_1, ..., [\text{MASK}], ..., w_n]$.
Dando in input $S\prime$ al modello BERT, esso associa ad ogni parola del proprio lessico la probabilità di corrispondere la parola mascherata. Le prime n parole con la probabilità più alta sono dette candidati. È quindi formalizzata come segue la funzione del BERT MLM detta "mask-filling":

$$B : S\prime \rightarrow [c_1, ..., c_n] \tag{4}$$

dove ogni $c_i \in [c_1, ..., c_n]$ è un candidato.

Esempio Data la frase

"che assistono ragazze in difficoltà, le persone soie e abbandonate, gli ammalati e gli anziani."

la parola *"soie"* sottolineata è stata individuata come errore. È quindi necessario mascherala, per dare la frase in input al modello BERT. La frase diventa dunque:

"che assistono ragazze in difficoltà, le persone [MASK] e abbandonate, gli ammalati e gli anziani."

BERT produce quindi una lista di candidati. Sono riportati i risultati impostando come soglia $n = 5$. I candidati rappresentano le top-5 più probabili correzioni.

- *"sole"* con probabilità 0.42
- *"anziane"* con probabilità 0.28
- *"povere"* con probabilità 0.08
- *"care"* con probabilità 0.03
- *"disabili"* con probabilità 0.01

Bisogna sottolineare come la parola originale sia trasparente al modello BERT. Ciò significa che i candidati prodotti dal modello sono del tutto indipendenti dalla parola originale, e sono inferite unicamente dal contesto derivato dal resto della frase.

4.2.3 Modulo di correzione Token

Error Detection

Il modulo di correzione token ha lo scopo di individuare e correggere tutti gli errori di tipo NW. L'individuazione degli errori è effettuata in modo automatico dopo una pre-elaborazione (pre-processing) del testo. Il funzionamento di questa fase può quindi essere suddiviso in due stadi:

1. **Sentence pre-processing**: la frase viene tokenizzata, ovvero viene divisa singole parole dette token. Questo stadio è implementato tramite la libreria NLTK[13].
2. **Error marking**: ogni token è analizzato singolarmente per determinare se contenga o meno un errore. Un token è considerato errato se sono vere le seguente condizioni:
    - Ha una lunghezza minima di 2 caratteri. Questa condizione è necessaria per evitare di introdurre nuovi errori correggendo inutilmente errori di tipo SP. Ad esempio, sarebbe deleterio provare a correggere le singole lettere di *"Q u a l c h e"*. Ciò è dovuto al fatto che il modulo di correzione token è progettato per correggere singoli token: nell'esempio precedente, si tenterebbe quindi la correzione di tutte le sub-word che compongono il token. Per questo tipo di errori, è stato sviluppato di modulo di correzione Split (sottosezione 4.2.4).

- Dato un vocabolario di parole corrette, il token considerato non è presente in tale vocabolario. La ricerca nel vocabolario è eseguita in maniera case-insensitive. Tale approccio, seppur molto semplice ha alcune criticità. Ad esempio, ogni parola non appartenente alla lingua italiana verrà segnalata come errore. È quindi auspicabile disporre di un vocabolario che comprenda anche alcune parole straniere di uso comune.

Un'eccezione a quest'ultima condizione è data da tutti i token che precedono un apostrofo. Questi token sono considerati corretti se almeno una fra tutte le combinazioni token + vocale è corretta. Si prenda ad esempio il token *"dell"* non presente nel vocabolario, seguito dal token apostrofo. Vengono generate le 5 combinazioni token-vocale *"della"*,*"delle"*,*"delli"*,*"dello"*,*"dellu"*. Siccome almeno una di queste è corretta, il token è considerato corretto.

Il processo appena descritto è schematizzato in Figura 6. Tutti gli errori identificati sono poi corretti in sequenza dalla fase di error correction.

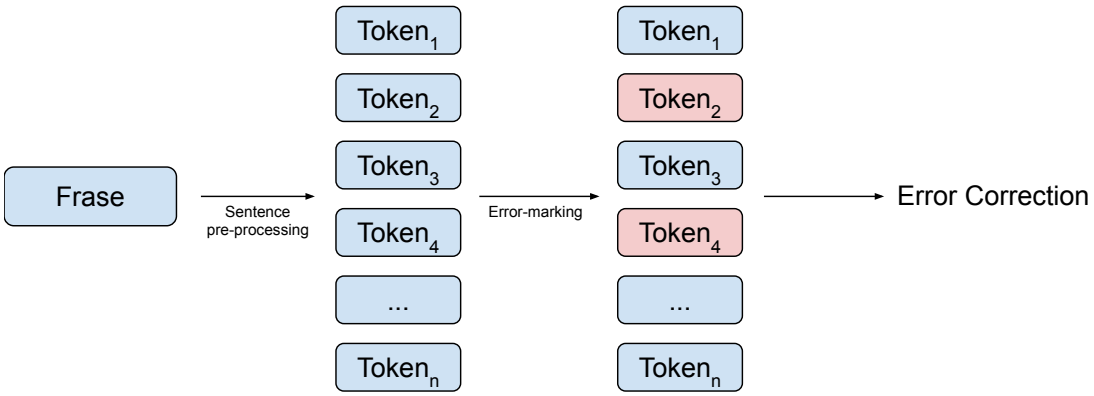

Figura 6: Schema del processo di error detection

Error Correction

La fase di error correction ha lo scopo di correggere gli errori individuati nella precedente fase. La fase di error correction è composta dai seguenti stadi:

1. Masking
2. Detokenization
3. Candidates generation and picking
4. Validation

Ognuno di questi stadi è ripetuto per ognuno degli errori identificati durante la fase precedente.

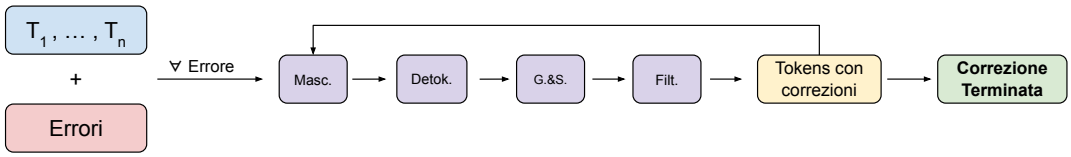

Figura 7: Schema della fase di error correction

Masking Lo stadio di masking ha lo scopo di mascherare uno dei token errati individuati nella fase precedente. Si ricorda che, a causa del funzionamento del BERT MLM descritto nella sottosezione 4.2.2, è possibile mascherare solo un token alla volta.
Data una sequenza di token $S = [t_1, ..., t_n]$ e l'insieme $E = [t_i, t_j, ...] \subseteq S$ dei token contenenti errore non ancora corretti, lo stadio di masking è descritto dalla seguente funzione:

$$M : S \rightarrow [t_1, ..., [\text{MASK}], ..., t_n] = S\prime \tag{5}$$

dove *[MASK]* sostituisce il primo token estratto dall'insieme E. Quanto descritto corrisponde al primo passaggio nella Figura 8.

Detokenization Come spiegato nella sottosezione 4.2.2, la funzione di mask-filling del BERT MLM richiede come input una sequenza intesa come frase, e non come insieme di token. È quindi compito dello stadio di detokenization ricomporre una sequenza a partire dall'output dello stadio precedente. Lo stadio di detokenization è descritto dalla seguente funzione:

$$D : [t_1, ..., [\text{MASK}], ..., t_n] \rightarrow F_{mask} \tag{6}$$

dove F_{mask} è una frase contente una maschera.

Candidates generation and picking Lo stadio di candidates generation and picking sfrutta il BERT MLM per generare dei candidati per la correzione, e sfrutta un'euristica per determinare il candidato con la maggior probabilità di essere corretto (da qui in poi riferito come soluzione). Più precisamente, la generazione dei candidati avviene tramite la funzione di mask-filling descritta nell'Equazione 4 (sottosezione 4.2.2). La fattibilità di applicare tale approccio per la generazione di

candidati per la correzione è dimostrata in [14]. Nel paper appena citato si riporta come, usando una combinazione di BERT e FastText, la giusta correzione da applicare sia presente fra i candidati prodotti con una probabilità del 70%. TODO: inserire riferimento ai risultati del capitolo analisi errore.
Lo step di generazione dei candidati è formalizzato come segue:

$$G : F_{mask} \rightarrow [c_1, ..., c_n] = C \tag{7}$$

dove ogni c_i è un candidato.
L'euristica per la scelta della soluzione sceglie il candidato con la più bassa distanza di Levenshtein dal token mascherato. Più formalmente, chiamando m il token mascherato, la soluzione s è definita come segue:

$$s = \underset{c_i \in C}{\operatorname{argmin}}\, d_{lev}(c_i, m) \tag{8}$$

In caso uno o più candidati abbiano la stessa distanza dal token mascherato, viene scelto quello con la maggior probabilità prodotta da BERT. TODO: giustificazione dell'euristica attraverso risultato presenti nel capitolo di analisi errore

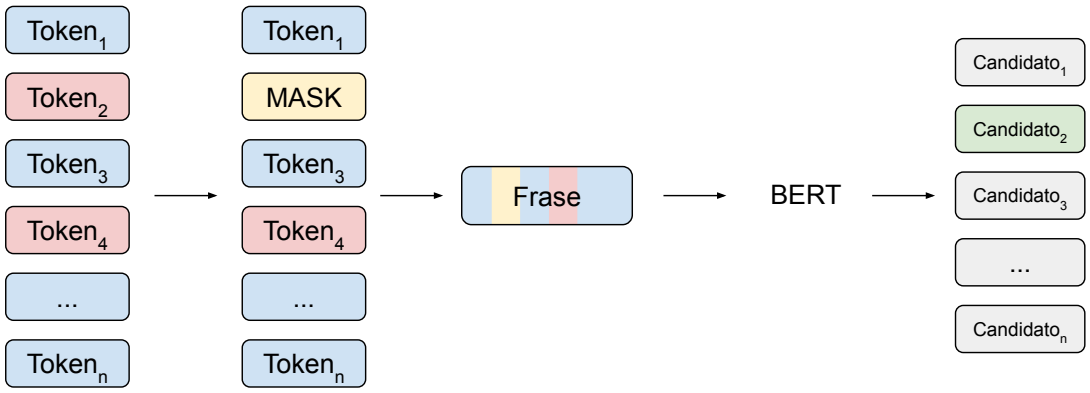

Figura 8: Schema del processo di generazione e scelta dei candidati

Validation Può accadere che la soluzione non sia una correzione adatta: si pensi ai seguenti casi:

1. L'error detection contrassegna un token corretto come errore. In questo caso, ogni tentativo di correzione introdurrebbe nuovi errori all'interno del testo.
2. BERT non produce la giusta correzione fra i candidati. Anche in questo caso, qualunque sia il candidato scelto, il sistema di correzione andrebbe a introdurre nuovi errori all'interno del testo.

È introdotta quindi un'ulteriore fase di validation, con lo scopo di valutare se la soluzione prodotta possa o meno rappresentare una correzione valida. La validazione è implementata tramite un'euristica che valuta se la soluzione trovata è troppo lontana (in termini di distanza di Levenshtein) dal token mascherato. Chiamando l la lunghezza in caratteri del token mascherato, e d la distanza di Levenshtein fra il token mascherato e la soluzione trovata, la soluzione è valida se:

- $l > 10 \wedge d < 5$
- $l > 5 \wedge d < 4$
- $l \leqslant 5 \wedge d < 3$

Se la validazione scarta la soluzione trovata il sistema ignora la correzione. Al contrario, se la validazione ritorna esito positivo, la correzione viene sostituita al token mascherato. Quanto appena descritto è rappresentato in Figura 9.

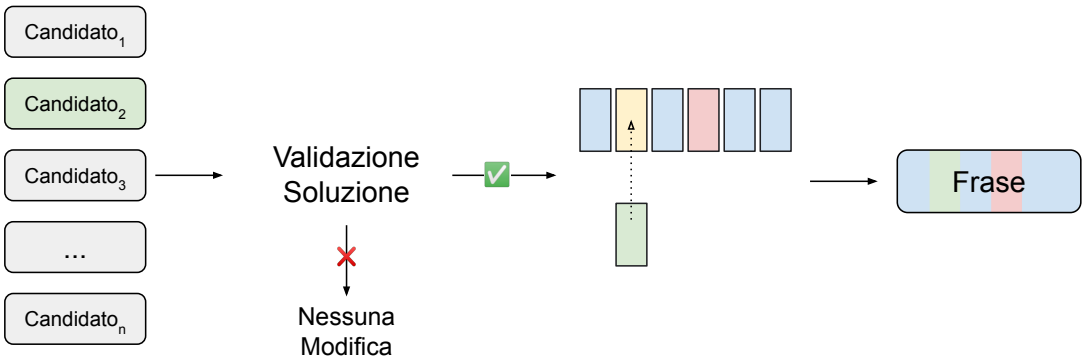

Figura 9: Schema del processo di scelta dei candidati

Il processo appena descritto si ripete per ogni errore contrassegnato durante la fase di error detection. Una volta completata la correzione dell'ultimo token errato, la fase di error correction può dirsi conclusa.

Ripetizione

In frasi contenenti molti errori, è possibile che una sola applicazione del modulo di correzione token non basti correggere tutti i non-word errors.
Ciò è dovuto al funzionamento di BERT, che usa il contesto a destra e sinistra della maschera per produrre i candidati di correzione. Se questo contesto è sporcato da molti errori, è più probabile che BERT non sia in grado di produrre una soluzione adeguata. È quindi possibile che dopo l'applicazione di questo o altri moduli, e

la conseguente correzione di alcuni errori, sia possibile ottenere migliori risultati sulle correzioni precedentemente non riuscite.
Questo approccio comporta però uno svantaggio: se la prima o una della prime correzioni sono sbagliate, il rischio è quello di introdurre rumore anche nelle correzioni successive.

4.2.4 Modulo di correzione Split

Error detection

Il modulo di correzione Split corregge tutti gli errori di tipo SP.
La fase di error detection si compone di tre stadi:

1. **Sequences marking**: data un frase contenente errori di tipo SP, sono delimitati i punti di inizio e terminazione delle sequenze contenenti errori. L'individuazione di tali sequenze è eseguita tramite la seguente espressione regolare:

   r ' (? : \ s | ^) (\ w (? : \W? \ s \ w { 1 , 2 }) { 3 , }) (? : \W| \ s | $) '

   La precedente espressione regolare intercetta tutte le sequenze di almeno tre gruppi di caratteri alfanumerici con lunghezza massima due intervallati da spazi e/o caratteri non alfanumerici. Per maggiore chiarezza, in Tabella 2 sono riportati alcuni esempi di sequenze riconosciute.

2. **Punctuation filtering**: all'interno delle sequenze vengono rimossi tutti i caratteri non alfanumerici o spaziature. Ciò serve e rimuovere eventuali segni di punteggiatura spuri che possono diminuire l'efficacia delle seguenti fasi.

3. **Error validation:** non tutte le sequenze riconosciute dall'espressione regolare sono degli errori. Si pensi ad esempio ad una serie di preposizioni o congiunzioni, come il quarto esempio in Tabella 2. Si usa quindi un'euristica per scartare le sequenze che si ritenga non contengano errori. Sono considerati errori tutte le sequenze nelle quali il numero di gruppi di 2 caratteri è minore del numero di gruppi di 1 carattere.

#	Frase	Sequenza	Errore
1	*"d i l e t t i membri, della ..."*	*d i l e t t i*	Sì
2	*"guidato d ui l l a fede"*	*d ui l l a*	Sì
3	*"... altro tipo di u n i o, n e."*	*u n i o, n e*	Sì
4	*"in me e io in te, siano anch..."*	*in me e io in te*	No
5	*"...dopo qu al ch e esitazione..."*	*qu al ch e*	Sì
6	*"...dopo q u al che esitazione..."*	*q u al*	Sì
7	*"...dopo qu al che esitazione..."*	/	/

Tabella 2: Esempi di sequenze riconosciute

L'approccio di error correction appena esposto presenta però alcune limitazioni. Se uno dei gruppi della parola divisa contiene più di tre caratteri alfanumerici, la sequenza prodotta può:

- Non essere riconosciuta. È il caso dell'esempio 7 nella Tabella 2. In questo caso, il gruppo *"che"* non è catturato dall'espressione regolare. Ciò comporta che il numero di gruppi di caratteri sia troppo basso per essere riconosciuto dall'espressione regolare.
- Essere riconosciuta parzialmente. È il caso dell'esempio 6 nella Tabella 2. In questo caso, il gruppo *"che"* non è catturato dall'espressione regolare. Ciò comporta che venga catturata la sequenza *"q u a l"*, che verrà corretta, introducendo possibilmente un errore. Il gruppo *"che"*, invece, viene erroneamente considerato corretto.

Data quindi una frase $f \in F$ contenente errori di segmentazione, dove F è l'insieme di tutte le frasi, la fase di error detection è formalizzata come segue:

$$ED_{split} : F \rightarrow L_{seq} \tag{9}$$

dove ogni $l_{seq} \in L_{seq}$ è un insieme di tuple $[(s_1, b_1, e_1), ..., (s_n, b_n, e_n)]$ nel quale ogni tupla corrisponde ad un errore individuato e validato. Ogni tupla è composta come segue:

- s_i è la sequenza filtrata contenente l'errore
- b_i è il punto di inizio in f della sequenza
- e_i è il punto di fine in f delle sequenze

Esempio Data la frase:

coinvolsero ogni uomo e o' g n i donna, p r o c u r a n d o.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
c	o	i	n	v	o	l	s	e	r	o		o	g	n	i		u	o	m

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
o		e		o	’		g		n		i		d	o	n	n	a	,	

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
p		r		o		c		u		r		a		n		d		o	.

Tabella 3: Caratteri e indici delle frase esempio

si procede con lo stadio di sequences marking. Le sequenze individuate sono quelle sottolineate nella frase data. Sono riportate di seguito insieme alla loro posizione di inizio e di fine.

#	**Sequenza**	**Pos.Inizio**	**Pos.Fine**
1	*”e o’ g n i”*	22	32
2	*”p r o c u r a n d o”*	40	59

Si procede quindi con lo stadio di punctuation filtering:

#	**Sequenza**	**Pos.Inizio**	**Pos.Fine**
1	*”e o g n i”*	22	32
2	*”p r o c u r a n d o”*	40	59

Si noti come nella prima sequenza è stato cancellato un apostrofo, ma le posizioni di inizio e fine rimangono le stesse, in quanto demarcano l’inizio e la fine della sequenza nella frase originale. Lo stadio di error validation in questo caso non scarta alcuna soluzione: in entrambe le sequenze, infatti, il numero di gruppi di 1 carattere è maggiore del numero di gruppi di 2 caratteri.

Error correction

La fase di Error Correction mira a correggere gli errori individuati nella fase precedente. Più formalmente è definita come:

$$EC_{split} : (F, L_{seq}) \rightarrow F_{corr} \tag{10}$$

dove F_{corr} è la frase con le correzione apportate.
La fase di Error Correction si compone dei seguenti stadi:

1. **Compression:** per ogni sequenza s_i individuata da correggere, si rimuovono gli spazi bianchi, in modo da ottenere un'unica parola continua detta $s\prime_i$.
2. **Vocabulary correction**: se una sequenza $s\prime_i$ è presente nel vocabolario, allora è $s\prime_i$ viene inserita nella frase al posto della sequenza contenente l'errore.
3. **BERT correction:** si fa uso del BERT MLM per trovare una correzione, in modo simile a quanto fatto nel modulo di correzione token. A seconda della tipologia dell' errore sono applicabili due strategie di correzione.
   - Con sottrazione
   - Con rimpiazzo

Ognuno degli stadi appena descritti è applicato in sequenza ad ogni singola tupla prodotta dalla fase precedente.

Funzione di sostituzione Viene introdotta una funzione detta funzione di sostituzione. Data una frase $f \in F$, una sequenza $s \in S$, una valore di inizio $b \in I$ e un valore di fine $e \in E$, la funzione è definita come:

$$Sos : (F, S, I, E) \rightarrow F\prime \tag{11}$$

dove $F\prime$ è la frase F in cui tutti i caratteri fra b ed e sono stati sostituiti da s.
Alternativamente, definendo come T la tupla (s, b, e) è possibile scrivere la funzione anche come segue:

$$Sos : (F, T) \rightarrow F\prime \tag{12}$$

Ad esempio, data la frase f_{es}:

Onorando una trod1z1one plurisecolare

e la tupla t_{es} (*"tradizione"*,14,25), si ha che:

$$Sos(f_{es}, t_{es}) = \textit{Onorando una tradizione plurisecolare} \tag{13}$$

Compression Lo stadio di compression rimuove gli spazi da ogni sequenza s individuata come errore. Più formalmente, data una tupla $(s, b, e) \in T$, lo stadio di compression è definito come segue:

$$Comp : T \rightarrow T\prime \tag{14}$$

dove in cui $T\prime$ è l'insieme di tutte le tuple $(s\prime, b, e)$ in cui $s\prime$ è la sequenza s senza spaziature. Riprendendo l'esempio proposto nella fase di error detection, le tuple prodotte diventano rispettivamente:

#	**Sequenza**	**Pos.Inizio**	**Pos.Fine**
1	*"eogni"*	22	32
2	*"procurando"*	40	59

Vocabulary correction Date le tuple prodotte dallo stadio precedente, lo stadio di vocabulary correction applica una correzione per tutte le tuple la cui sequenza corrisponde ad una parola presente nel vocabolario.
Data quindi la frase F contenente errori di segmentazione e una tupla $(s\prime, b, e)$ detta t_{corr} che rispetta la condizione enunciata precedentemente, la frase con la correzione applicata F_{corr} si ottiene come segue:

$$F_{corr} = Sos(F, t_{corr}) \tag{15}$$

Se la tupla che si sta trattando può essere utilizzata per una correzione con vocabolario, la correzione si arresta in questo stadio. Altrimenti, si procede allo stadio di BERT correction.
Continuando l'esempio predente, si nota come la seconda tupla contenga la sequenza "procurando" che è una parola all'interno del vocabolario. Ricordando la frase originale

coinvolsero ogni uomo e o' g n i donna, p r o c u r a n d o.

detta F_{ex}, e la tupla t_2 (*"procurando"*, 40, 59), la correzione si applica come segue:

$$F_{corr} = Sos(F_{ex}, t_2) \tag{16}$$

Il risulta è la frase F_{corr}:

coinvolsero ogni uomo e o' g n i donna, procurando.

BERT correction Per tutte le tuple per le quali non è stato possibile applicare una correzione con vocabolario, si passa allo stadio di BERT correction. Il primo

step di questo stadio consiste nel mascheramento della sequenza errata nella fase originale. Data una tupla $(s\prime, b, e)$ prodotta dallo stadio di compressione e una frase F, si ottiene la frase mascherata F_{mask} come segue:

$$F_{mask} = Sos(F, [MASK], b, e) \tag{17}$$

A questo punto si sfrutta il BERT MLM per produrre un candidato per la correzione. Il processo è analogo alla fase di generazione e scelta dei candidati del modulo di correzione token nella sezione 4.2.3, per cui si rimanda il lettore al paragrafo che ne spiega il funzionamento.
Una volta prodotto il candidato per la correzione, detto c_{corr}, è possibile applicare una correzione per sottrazione o per rimpiazzo.

Correzione con sottrazione Si applica in caso c_{corr} coincida con la fine o l'inizio della sequenza compressa $s\prime$. Questo è il caso in cui la sequenza riconosciuta come errore di spezzettamento con spazi comprende due parole. Pertanto, se c_{corr} corrisponde all'inizio di $s\prime$, si ottengono le due stringhe sub_1 e sub_2 come segue:

$$sub_1, sub_2 = c_{corr}, s\prime - c_{corr} \tag{18}$$

Se invece c_{corr} corrisponde alla fine di $s\prime$, le stringhe sub_1 e sub_2 si ottengono come segue:

$$sub_1, sub_2 = s\prime - c_{corr}, c_{corr}, \tag{19}$$

Data quindi la frase F contenente errori di segmentazione, la tupla $(s\prime, b, e)$ e le stringhe sub_1, sub_2, la frase con la correzione applicata F_{corr} si ottiene come segue:

$$F_{corr} = Sos(F, sub_1 + \text{" "} + sub_2, b, e) \tag{20}$$

La correzione con sottrazione corregge i casi in cui la sequenza contenente errore è formata da due parole. Può accadere che la stringa $s\prime - c_{corr}$ non rappresenti una parola corretta, ovvero potrebbe non essere presente nel vocabolario. Non sono previsti controlli e contromisure ad hoc per tale eventualità: la correzione di eventuali errori di questo tipo è delegata ai moduli di correzione token che possono essere inseriti a valle nella pipeline.
Si consideri ora l'esempio portato avanti negli stadi precedenti: data la tupla t_2 (*"eogni"*, 22, 32) e la frase F_{ex}

coinvolsero ogni uomo e o' g n i donna, p r o c u r a n d o.

si ottiene come la frase F_{mask} applicando la funzione di sostituzione:

$$F_{mask} = Sos(F_{ex}, [MASK], 22, 32) \tag{21}$$

Si ottiene dunque la frase

coinvolsero ogni uomo [MASK] donna, p r o c u r a n d o.

Sfruttando il BERT MLM si ottiene il candidato c_{corr} "ogni". Il candidato corrisponde alla fine di $s\prime$, pertanto si ottengono $sub_1, sub_2 = e, ogni$. Si applica quindi la funzione di sostituzione:

$$F_{corr} = Sos(F_{ex}, e + " " + ogni, 22, 32) \tag{22}$$

La frase corretta F_{corr} è quindi:

coinvolsero ogni uomo e ogni donna, p r o c u r a n d o.

Correzione con rimpiazzo Si applica in caso c_{corr} non coincida con la fine o l'inizio della sequenza compressa $s\prime$. Questo step utilizza un'euristica per validare la correzione prodotta dal BERT MLM. La soluzione prodotta da BERT è valutata ammissibile e solo se $d_{lev}(s\prime, c_{corr}) \leqslant 4$.
In caso la soluzione sia ammissibile, la correzione è applicata tramite la funzione di sostituzione:

$$F_{corr} = Sos(F, c_{corr}, b, r) \tag{23}$$

In caso la soluzione non sia ammissibile, la correzione si arresta senza apportare correzioni.

Capitolo 5

Implementazione, Test e Risultati

Capitolo 6

Analisi dell'errore

Bibliografia

[1] Adam Jatowt, Mickael Coustaty, Nhu-Van Nguyen, Antoine Doucet, et al. Deep statistical analysis of ocr errors for effective post-ocr processing. In *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 29–38. IEEE, 2019.

[2] Roger T Hartley and Kathleen Crumpton. Quality of ocr for degraded text images. *arXiv preprint cs/9902009*, 1999.

[3] Guillaume Chiron, Antoine Doucet, Mickaël Coustaty, Muriel Visani, and Jean-Philippe Moreux. Impact of ocr errors on the use of digital libraries: towards a better access to information. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–4. IEEE, 2017.

[4] Stephen Mutuvi, Antoine Doucet, Moses Odeo, and Adam Jatowt. Evaluating the impact of ocr errors on topic modeling. In *International Conference on Asian Digital Libraries*, pages 3–14. Springer, 2018.

[5] Rose Holley. How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs. *D-Lib Magazine*, 15(3/4), 2009.

[6] Wojciech Bieniecki, Szymon Grabowski, and Wojciech Rozenberg. Image preprocessing for improving ocr accuracy. In *2007 international conference on perspective technologies and methods in MEMS design*, pages 75–80. IEEE, 2007.

[7] Yaakov HaCohen-Kerner. What is differance between string and token in natural language processing techniques?, 06 2014.

[8] Wikipedia. N-gramma — wikipedia, l'enciclopedia libera, 2020. [Online; in data 3-novembre-2021].

[9] Thorsten Brants and Alex Franz. Web 1t 5-gram version 1, set 2006.

[10] Youssef Bassil and Mohammad Alwani. Ocr context-sensitive error correction based on google web 1t 5-gram data set. *arXiv preprint arXiv:1204.0188*, 2012.

[11] Andrew Carlson and Ian Fette. Memory-based context-sensitive spelling correction at web scale. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 166–171. IEEE, 2007.

[12] Kevin Atkinson. Gnu aspell, 1998. *Software available at http://aspell. net.*

[13] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[14] Mahdi Hajiali, Jorge Ramón Fonseca Cacho, and Kazem Taghva. Generating correction candidates for ocr errors using bert language model and fasttext subword embeddings. In *Intelligent Computing*, pages 1045–1053. Springer, 2022.