Scadenze:
Presentazione domanda e titolo: entro il 22 Novembre
Consegna riassunto: dal 22 novembre al 6 dicembre
Consegna tesi: entro il 6 dicembre

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

Capitolo 3

Dataset e Perturbazione

Capitolo 4

Metodologia

In questo capitolo è descritta la metodologia del sistema 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, e sono le fasi e le componenti del sistema. Nella Equazione 3 è descritto il funzionamento del BERT MLM, che sta alla base del processo di correzione. 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 del testo da un'immagine. Più precisamente, per digitalizzare si intende ottenere del testo in una una codifica leggibile da una macchina, come ad esempio ASCII o Unicode.

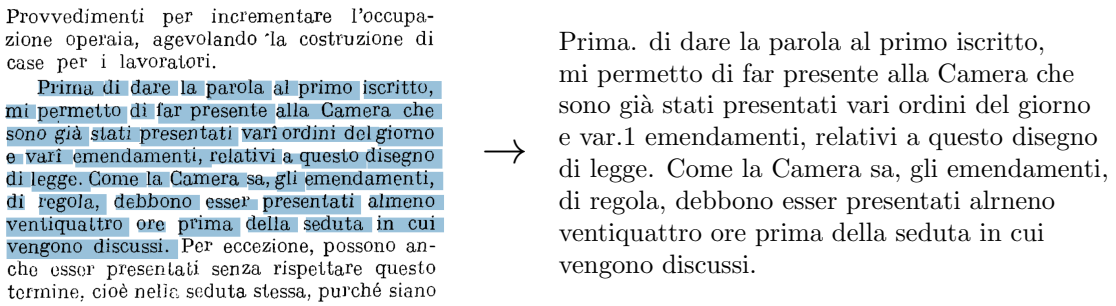

Figura 1: Esempio di testo estratto tramite OCR

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 [1][2]. Si veda, ad esempio, il testo estratto in Figura 1: 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:

- **Errore di token**. Si ha un errore di token quando aggiunte, rimozioni o sostituzioni di caratteri occorrono all'interno di una parola senza però dividerla o eliminarla. Questo tipo di errore si divide in due ulteriori sottocategorie:
  - Non-word error (NW), quando la parola risultante non è presente in un dato vocabolario.
  - Real-word error (RW), quando la parola risultante è presente in un dato vocabolario.
- **Errore di segmentazione**. Si ha un errore di segmentazione quando aggiunte, rimozioni o sostituzioni di caratteri 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:
  - Spezzettamento con spazi (SP), quando i caratteri di una parola o più parole adiacenti sono intervallati da spaziature.
  - Divisione con punteggiatura (DP), quando un segno di punteggiatura divide in due una parola.
  - Punteggiatura spuria (PS), quando un segno di punteggiatura viene aggiunto fra una parola e l'altra.

In Tabella 1 sono riportati alcuni esempi di errori.

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 di più deteriorati, come archivi storici. Per lo stesso motivo l'intensità degli errori può variare a seconda delle specifiche condizioni delle specifiche parti del documento.

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: la parola *"amati"* viene riconosciuta come *"amolfi"*. Il sistema potrebbe correggere con *"amati"*, introducendo un errore.

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 i real-word error (RW) e gli errori di spezzettamento con spazi (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:

$$M : F \mapsto F\prime \tag{1}$$

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.

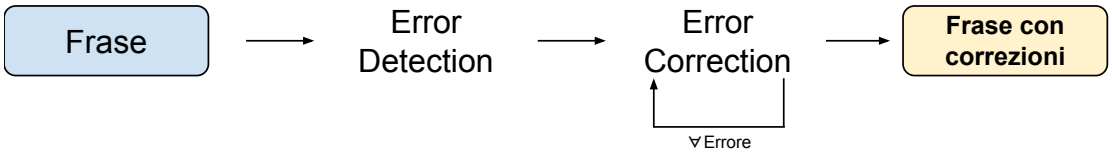

Figura 2: Schema del funzionamento di un modulo

Come si può vedere in Figura 2, 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:

$$P : f_1 \circ f_2 \circ ... \circ f_n \tag{2}$$

dove ogni f_i è un modulo.

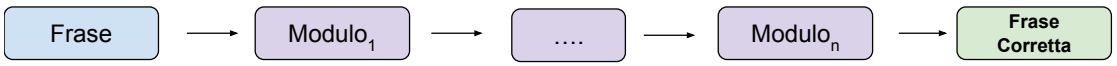

Figura 3: Schema riassuntivo della pipeline di correzione

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.

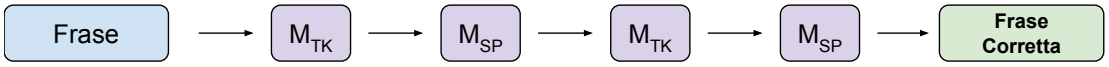

Figura 4: Esempio di una possibile pipeline con più occorrenze di uno stesso modulo

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":

$$B : S\prime \rightarrow [c_1, ..., c_n] \tag{3}$$

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, di cui sono riportati solo i primi 5:

- *"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 i non-word error (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. **Pre-elaborazione**: la frase viene tokenizzata, ovvero viene divisa singole parole dette token. Questo stadio è implementato tramite la libreria NLTK[3].

2. **Identificazione degli errori**: 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"*.
   - 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.

   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 5. Tutti gli errori identificati sono poi corretti in sequenza dalla fase di error correction.

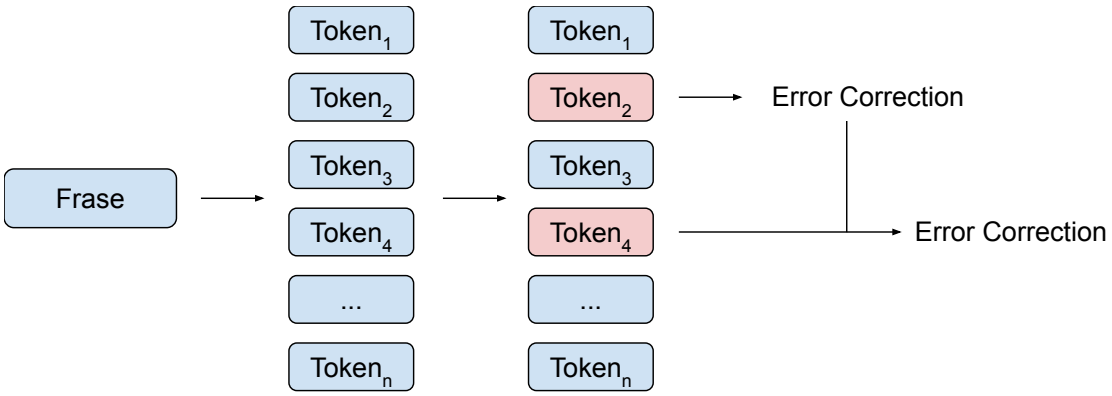

Figura 5: Schema del processo di error detection

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. Mascheramento
2. Detokenizzazione
3. Generazione e scelta dei candidati
4. Validazione

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

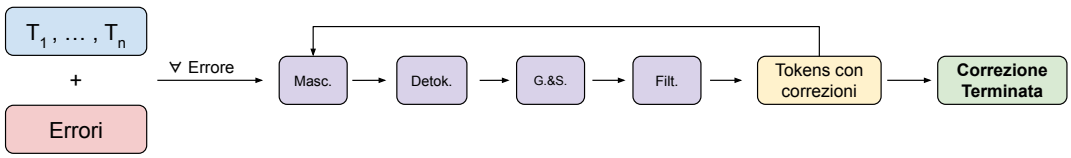

Figura 6: Schema della fase di error correction

Mascheramento Lo stadio di mascheramento 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 mascheramento è descritto dalla seguente funzione:

$$M : S \rightarrow [t_1, ..., [\text{MASK}], ..., t_n] = S\prime \tag{4}$$

dove *[MASK]* sostituisce il primo token estratto dall'insieme E. Quanto descritto corrisponde al primo passaggio nella Figura 7.

Detokenizzazione 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 detokenizzazione ricomporre una sequenza a partire dall'output dello stadio precedente. Lo stadio di detokenizzazione è descritto dalla seguente funzione:

$$D : [t_1, ..., [\text{MASK}], ..., t_n] \rightarrow F_{mask} \tag{5}$$

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

Generazione e scelta dei candidati Lo stadio di generazione e scelta dei candidati 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 3 (sottosezione 4.2.2). La fattibilità di applicare tale approccio per la generazione di candidati per la correzione è dimostrata in [4]. 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:

$$G : F_{mask} \rightarrow [c_1, ..., c_n] = C \tag{6}$$

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:

$$s = \underset{c_i \in C}{\operatorname{argmin}}\, d_{lev}(c_i, m) \tag{7}$$

In caso uno o più candidati abbiamo 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

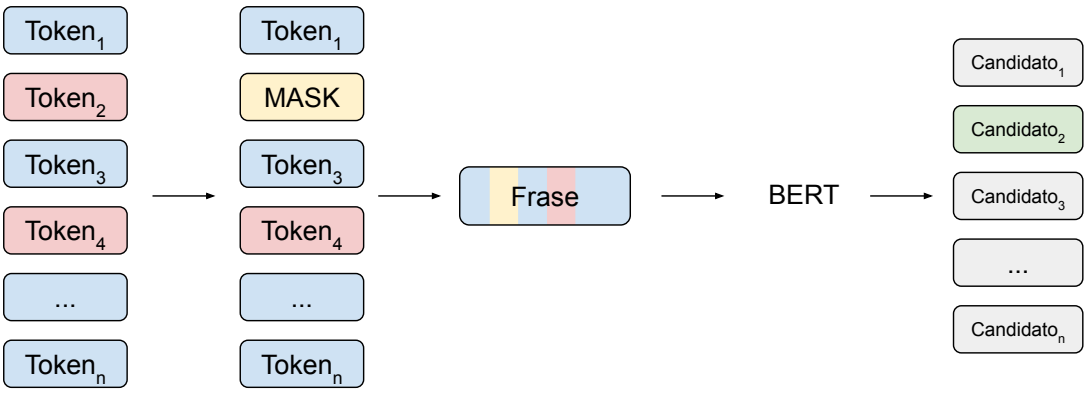

Figura 7: Schema del processo di generazione e scelta dei candidati

Validazione 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 validazione, 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 8.

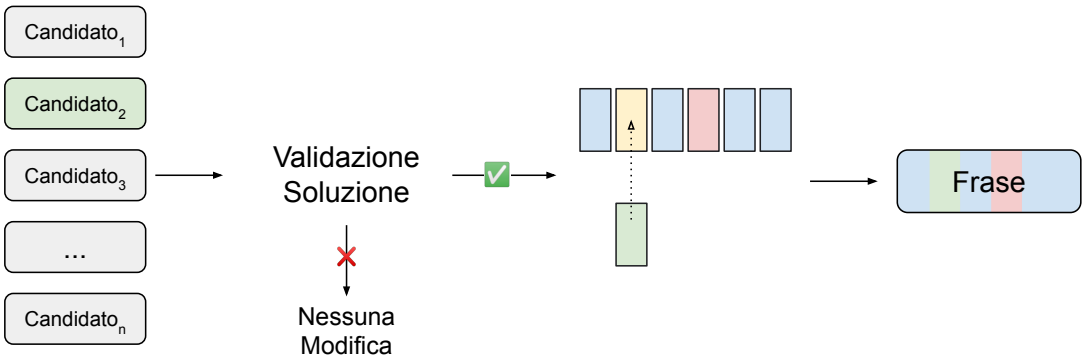

Figura 8: Schema del processo di scelta dei candidati

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.

4.2.4 Modulo di correzione Split

Capitolo 5

Implementazione, Test e Risultati

Capitolo 6

Analisi dell'errore

Bibliografia

[1] 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.

[2] 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.

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

[4] 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.