Correttore BERT

Sebastiano Caccaro

16 settembre 2021

Sommario

In questo documento è descritto il correttore sviluppato sulla base di un modello BERT

1 BERT

BERT[1] (Bidirectional Encoder Representations from Transformers) è una tecnica di machine learing basata sui transformer utilizzata nel campo del NLP (Natural Language Processing) per il pre-training di modelli linguistici.

1.1 Modello

Allenare da zero un modello è un processo altamente dispendioso. Per la nostra applicazione è stato quindi deciso di utilizzare un modello pre-allenato su dei corpora in lingua italiana, chiamato *dbmdz/bert-base-italian-xxl-uncased*[2].

1.2 Libreria

La libreria Transformers[3] fornisce una API unificata per utilizzare diverse architetture di transformer, fra le quali anche BERT.
La libreria mette a disposizione una serie di task. In particolare, per la nostra applicazione, verrà utilizzato "Fill Mask".

Per utilizzare questa funzione, è innanzitutto necessario mascherare una parola all'interno di un frase. Per fare ciò, basta sostituire *"[MASK]"* alla parola scelta. Dando questa frase in input al sistema, verranno generate della parole candidate al posto della maschera,

insieme alla probabilità che ogni parola sia quella corretta. Ognuna delle parole generate è scelta in base al contesto della frase.
È inoltre importante sottolineare come per una frase non sia possibile mascherare più di una parola.
Ad esempio, partendo dalla frase

"Roma è la capitale d'Italia."

è possibile mascherare *"capitale"* e ottenere la seguente frase:

"Roma è la [MASK] d'Italia."

Il sistema genera quindi dei candidati per sostituire la maschera:

1. *"capitale"* con probabilità 0.906
2. *"città"* con probabilità 0.030
3. *"storia"* con probabilità 0.009
4. *"regina"* con probabilità 0.005
5. *"culla"* con probabilità 0.005

È chiaro come in questo caso l'opzione 1 sia quella corretta.

2 Sistema di correzione

2.1 Funzionamento generale

Data una qualsiasi frase contenente degli errori in input, lo scopo del sistema di correzione è quello di fornire in output la stessa frase senza gli errori. A tale scopo è necessario:

- Individuare gli errori
- Correggere gli errori individuati
- Non introdurre nuovi errori

Il sistema di correzione è organizzato come una pipeline, la quale può essere composta da più moduli. Ogni modulo può essere ripetuto una o più volte. Concettualmente, ogni modulo è modellato per individuare e correggere uno specifico tipo di errore.
Più formalmente un modulo può essere definito come la seguente funzione:

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

dove T e $T\prime$ sono rispettivamente la stringa di testo originale e quella corretta.
È quindi possibile definire una pipeline di correzione come segue:

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

dove ogni f_i è un modulo.

2.2 Modulo di Correzione Token

Il modulo di correzione token individua e corregge in ogni frase gli errori a livello di parola. Sono considerati errori solamente i cosiddetti non-word errors, ovvero quelle parole che non sono presenti nel vocabolario utilizzato per la correzione.
Il modulo agisce nelle seguenti fasi:

1. Tokenizzazione
2. Individuazione degli errori
3. Correzione degli errori
4. Detokenizzazione

2.2.1 Tokenizzazione

In questa fase lo scopo è quello di dividere una frase nelle parole che la compongono, ad esempio:

"I seminaristi hanno bisogno dell'attenzione personale"

diventa:

"I", "seminaristi", "hanno", "bisogno", "dell", "'", "attenzione", "personale"

Per svolgere questo task è utilizzata la libreria python NLTK[4] (The Natural Language Toolkit), che mette a disposizione una funzione apposita. L'output di questa funzione è inoltre ulteriormente processato per evitare che il carattere "'" (apostrofo singolo) venga considerato come parte di un token. È infatti fondamentale per le fasi successive che i simboli di punteggiatura siano separati dalle parole vere e proprie.

2.2.2 Individuazione degli errori

In questa fase lo scopo è quello di individuare i token contenti degli errori, che verranno poi sottoposti a correzione.
Dato un dizionario di parole, un token viene considerato errato se non è presente all'interno del dizionario. Sono però definite delle eccezioni a questa regola:

- Tutti i token di un carattere sono considerati corretti. Questo approccio permette di evitare di provare a correggere parole frammentate o segni di punteggiatura non presenti nel vocabolario.
- Se un token non è presente nel vocabolario ed è seguito da un token di apostrofo, è probabile che sia avvenuta l'elisione di una vocale. In questo caso, vengono generate cinque versioni diverse della stessa parola, ognuna terminante con una vocale diversa. Se almeno una di queste è presente nel dizionario, il token è considerato corretto.
  Nell'esempio proposto in precedenza si nota la sequenza *"dell", "'", "attenzione"*. Il token *"dell"* potrebbe non essere presente nel dizionario. Si verifica quindi se almeno uno fra *"della","delle","delli","dello","dellu"* è presente nel dizionario. In questo caso, quindi, il token *"dell"* è considerato corretto.

2.2.3 Correzione degli errori

Una volta che una parola viene segnala come errata è necessario tentare di correggerla. Il processo ha inizio detokenizzando, ovvero ricomponendo, la frase originaria. Nel farlo, si sostituisce *"[MASK]"* al token errato.
La detokenizzazione è implementata a partire dal modulo *TreebankWordDetokenizer* della libreria NLTK. L'output di questo modulo viene successivamente ulteriormente processato per rimuovere caratteristiche indesiderate, come degli spazi prima di alcuni segni di punteggiatura.
A questo punto, il sistema descritto nella sottosezione 1.2 genera i possibili token candidati. Il candidato scelto è quello che minimizza la distanza di Levenshtein fra sé stesso e il token errato: non viene fatto uso della probabilità prodotta dal sistema.
Può accadere che anche il migliore dei candidati prodotti possa non rappresentare una correzione plausibile. Ciò si verifica quando la di-

stanza di Levenshtein fra il token errato e il candidato supera una certa soglia. In questi casi è preferibile non correggere l'errore.

2.2.4 Detokenizzazione

A questo punto, se uno o più token all'interno della frase sono stati corretti, la frase viene detokenizzata e restituita. Altrimenti, viene restituita la frase originale.

2.3 Modulo di Correzione Split

Il modulo di correzione di split serve a correggere tutti quelli errori che risultano nella frammentazione di una o più parole adiacenti. Un esempio di tale tipo di errore è la frase:

"coinvolsero ogni uomo e ogni donna, procurando."

che diventa

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

È chiaro da questo esempio come le parole *ogni* e *procurando* siano state frammentate.
Il funzionamento del modulo può essere diviso nelle seguenti fasi:

- Individuazione degli errori
- Correzione degli errori

2.3.1 Individuazione degli errori

Gli errori di frammentazione vengono individuati dalla seguente espressione regolare:

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

La precedente espressione regolare intercetta tutte le sequenze di una o più lettere e numeri seguite da al più un carattere alfanumerico e intervallate da almeno due spazi.
Ognuna delle sequenze riconosciute viene sottoposta ad un ulteriore controllo per capire se vada o meno corretta. Nelle sequenze marcate come errore vengono rimossi tutti i caratteri non alfanumerici. Se il numero gruppi di caratteri rimanenti di lunghezza 2 supera il numero di gruppi di caratteri di lunghezza 1, la sequenza viene considerata corretta. Questo ulteriore controllo serve per evitare di correggere

erroneamente sequenze di articoli e/o preposizioni. Nella Tabella 1 sono riportati alcuni esempi di errori riconosciuti.

Frase	**Sequenza**	**Errore**
"d i l e t t i membri, della ..."	*d i l e t t i*	Sì
"guidato d ui l l a fede"	*d ui l l a*	Sì
"... altro tipo di u n i o, n e."	*u n i o, n e*	Sì
"in me e io in te, siano anch..."	*in me e io in te*	No

Tabella 1: Esempi di sequenze riconosciute

Per tutte le sequenze contrassegnate come errore, si provano i seguenti metodi:

1. Correzione con vocabolario
2. Correzione con BERT
    - Correzione Totale
    - Correzione con Sottrazione

La correzione con vocabolario è concettualmente semplice. Si rimuovono tutti i caratteri non alfanumerici dalla sequenza riconosciuta, e si controlla se questa nuova stringa (stringa ridotta da qui in poi) è o meno presente nel vocabolario. Se lo è, essa viene inserita nella frase al posto della sequenza riconosciuta.
Ad esempio, degli esempi in Tabella 1, *"d i l e t t i"* e *"u n i o, n e"* possono essere corretti con questo metodo.
Nel caso la stringa ridotta non sia nel vocabolario, come nel caso del secondo esempio di Tabella 1 (*"d ui l l a"*), si utilizza un altro metodo. In questo caso la correzione è molto simile a quanto avviene nel modulo di correzione token: viene mascherata la sequenza errata e si procede a generare dei candidati per la sostituzione. Viene quindi scelto il candidato più vicino alla stringa ridotta. Come per il modulo di correzione token, se nessuno dei candidati ha una distanza sufficientemente bassa dalla stringa ridotta, la frase originale è lasciata invariata.
È possibile che nella sequenza frammentata sia presente più di una parola. Questo accade quando sono presenti più parole frammentate adiacenti o una parola frammentata accanto ad parola di lunghezza inferiore a due. Un esempio è la frase:

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

dove la sequenza riconosciuta *"e o g n i"* è chiaramente formata da due parole distinte. Per ovviare a questo problema la parola suggerita dal sistema descritto in precedenza viene sottratta, se possibile, alla stringa ridotta.
Nell'esempio considerato viene suggerita la parola *"ogni"*. Si ottengono quindi due parole, *"e"* e *"ogni"*, che vengono sostituite nella frase. Il risultato finale è quindi:

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

Nell'esempio considerato il risultato della sottrazione fra le due stringhe (*"e"*) è un parola considerata corretta, perché presente nel dizionario. Questo è solo un caso, e il token risultante potrebbe anche essere errato. Questa eventualità è stata considerata nella costruzione della pipeline di correzione, e si può risolvere ripetendo una o più volte il modulo di correzione token.

2.4 Configurazione

Il sistema di correzione può essere composto da uno o più moduli ripetuti. I risultati sperimentali hanno portato alla seguente configurazione finale:

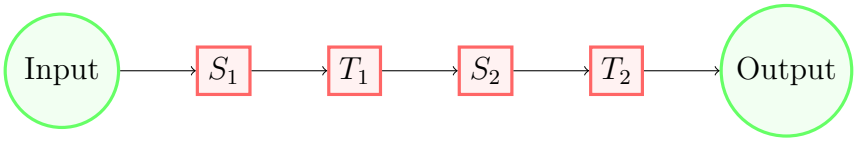

Figura 1: Configurazione della pipeline di correzione

Nello schema in Figura 1, S_i sta per modulo di correzione di split, mentre T_i sta per modulo di correzione token.
I risultati sperimentali hanno mostrato come ripetere due volte i moduli porti a un tangibile incremento delle performance di correzione, mentre una terza ripetizione si è dimostrata superflua. Ciò è dovuto ai seguenti motivi:

- Il modulo di correzione split può lasciare nel testo alcune parole non corrette a seguito di una correzione con sottrazione. Si pensi, ad esempio alla sequenza *"d ul v e r d e"*: il modulo di correzione potrebbe suggerire la parola *"verde"* e correggere quindi con *"dul verde"*. Questo procedimento lascia una parola errata che può

essere corretta con una o più passate del modulo di correzione token.

- Il modulo di correzione token salta le parole per le quali non riesce a trovare un candidato abbastanza vicino. È quindi possibile che in una frase con più correzioni necessarie la prima venga saltata, ma la seconda venga corretta. La seconda correzione potrebbe quindi fornire al modulo di correzione il contesto necessario per produrre candidati corretti in una seconda passata.

3 Risultati

Di seguito sono riportati i risultati sperimentali del sistema di correzione, secondo le metriche già descritte nel documento di valutazione. I risultati sono confrontati con quelli ottenuti dal metodo Pgp (Project Gender Politics) sullo stesso dataset.
Sono riepilogate brevemente le varie metriche:

1. Rapporto fra errori che sono stati corretti ed errori presenti nelle frasi perturbate;
2. Errori erroneamente introdotti dal processo di correzione per ogni singola frase;
3. Rapporto fra gli errori erroneamente introdotti dal processo di correzione e gli errori corretti dallo stesso processo.

	Metrica 1		**Metrica 2**		**Metrica 3**	
	Bert	Pgp	Bert	Pgp	Bert	Pgp
T1	0.33	0.14	0.21	1.22	0.4	5.44
T2	0.34	0.14	0.22	1.22	0.32	4.18
T3	0.29	0.14	0.25	1.21	0.28	2.84
S1	0.26	0.15	0.19	1.18	1.02	11.01
S2	0.25	0.16	0.19	1.16	0.89	8.09
S3	0.27	0.15	0.19	1.09	0.45	4.33
M1	0.29	0.14	0.22	1.16	0.36	3.83
M2	0.28	0.14	0.24	1.14	0.32	2.82
M3	0.23	0.13	0.28	1.11	0.29	1.93

Tabella 2: Risultati sperimentali del correttore BERT e del correttore Pgp

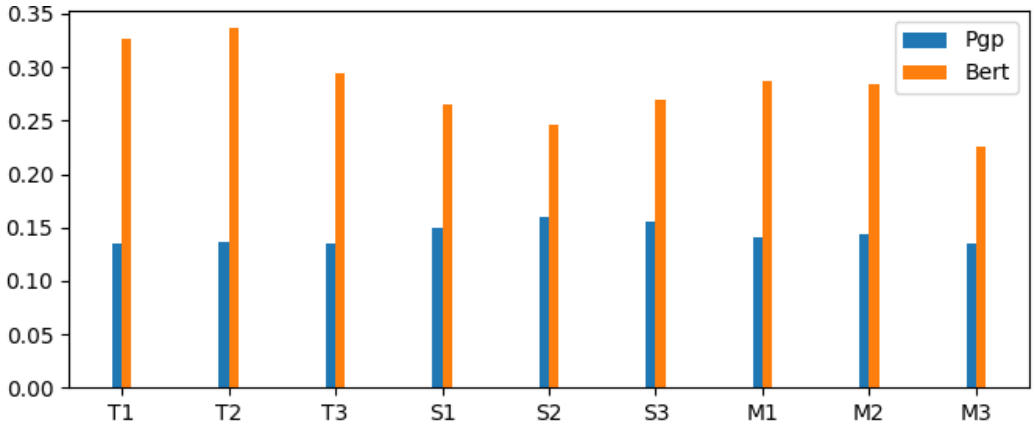

Figura 2: Rapporto fra errori che sono stati corretti ed errori presenti nelle frasi perturbate

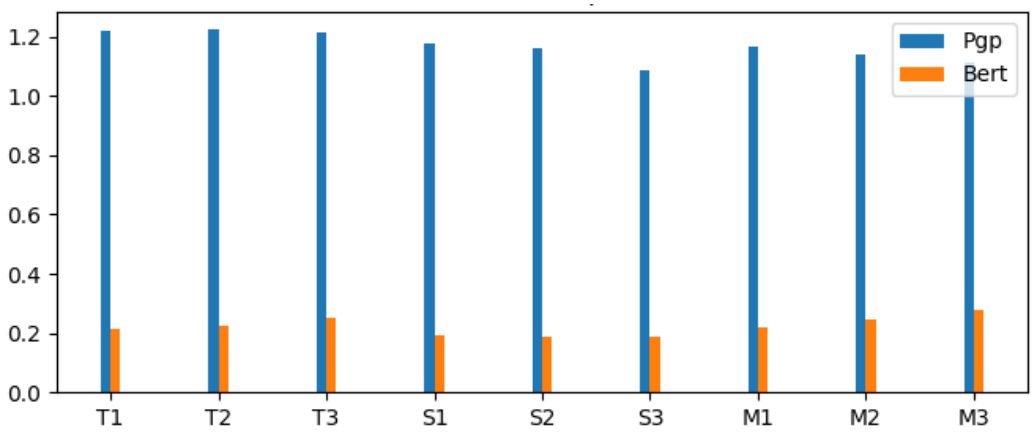

Figura 3: Errori erroneamente introdotti dal processo di correzione per ogni singola frase

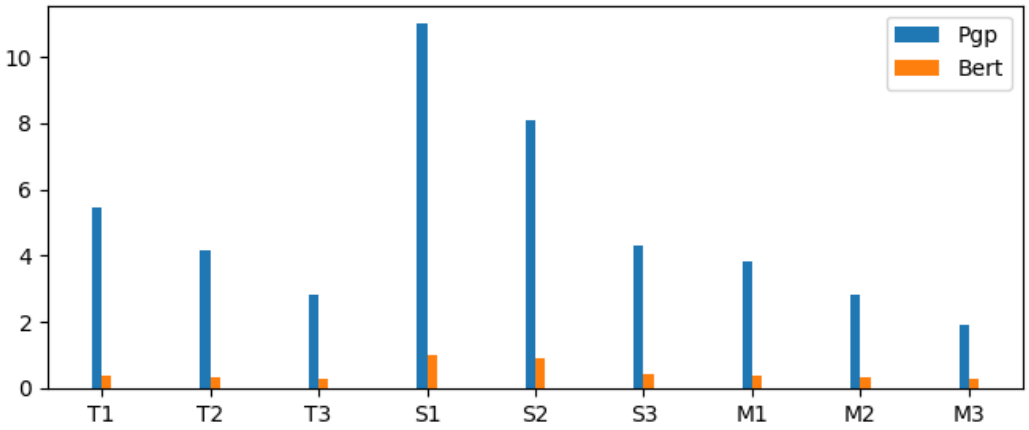

Figura 4: Rapporto fra gli errori erroneamente introdotti dal processo di correzione e gli errori corretti dallo stesso processo

I risultati mostrano come il correttore sviluppato presenti buone prestazioni, sopratutto nelle pipeline con solo errori di tokenizzazione. È particolarmente basso il numero di errori introdotti dal processo di correzione. Questo risultato è dovuto al modo in cui il sistema genera i candidati per la correzione. Essi infatti sono generati a partire dal contesto della frase e senza conoscere la parola da correggere. Ciò comporta che è possibile usare la distanza di Levenshtein per discriminare le varie possibilità, e lasciare la frase inalterata in caso, ad esempio, di falsi positivi.

3.1 Considerazioni e Osservazioni

Seguono alcune considerazioni sul sistema di correzione:

- La complessità di un'architettura come BERT comporta che il sistema sia abbastanza lento. Sicuramente utilizzare una macchina con dei CUDA cores darebbe uno speedup notevole, ma in ogni in caso il sistema è decisamente più lento di Pgp. Per queste ragioni, tutti i test sono stati eseguiti su una versione ridotta del dataset, composta da 10000 frasi scelte casualmente.
- Il sistema potrebbe essere ulteriormente esteso per migliorarne le performance, specialmente per gli errori di segmentazione. Al momento, infatti non è stato possibile trovare un metodo efficace per la correzione dei segni di punteggiatura errati.

- Nel dataset preso in considerazione, tutte le frasi hanno una lunghezza inferiore ai 50 caratteri. È lecito pensare, dato il funzionamento del sistema di correzione, che ad una maggior lunghezza corrispondano risultati migliori.

4 Risultati v2

4.1 Risultati con lunghezza raddoppiata

Vengono in seguito riportati i risultati di ulteriori test, svolti con una versione del dataset che contiene frasi di 100 caratteri di lunghezza.

	Metrica 1		**Metrica 2**		**Metrica 3**	
	Bert	Pgp	Bert	Pgp	Bert	Pgp
T1	0.4	0.24	0.41	2.17	0.32	2.8
T2	0.4	0.27	0.42	2.15	0.26	1.96
T3	0.37	0.28	0.48	2.21	0.22	1.36
S1	0.27	0.15	0.36	2.0	0.91	9.38
S2	0.25	0.17	0.36	1.96	0.83	6.4
S3	0.28	0.16	0.39	1.8	0.44	3.53
M1	0.32	0.22	0.47	2.08	0.35	2.3
M2	0.33	0.25	0.47	2.0	0.26	1.49
M3	0.26	0.24	0.65	2.04	0.3	1.05

Tabella 3: Risultati sperimentali del correttore BERT e del correttore Pgp con frasi di lunghezza 100

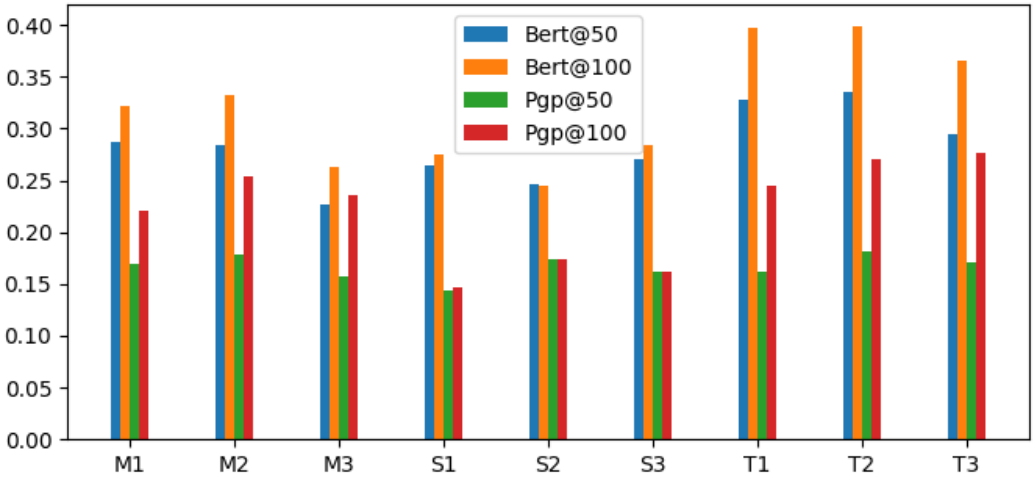

Figura 5: Rapporto fra errori che sono stati corretti ed errori presenti nelle frasi perturbate

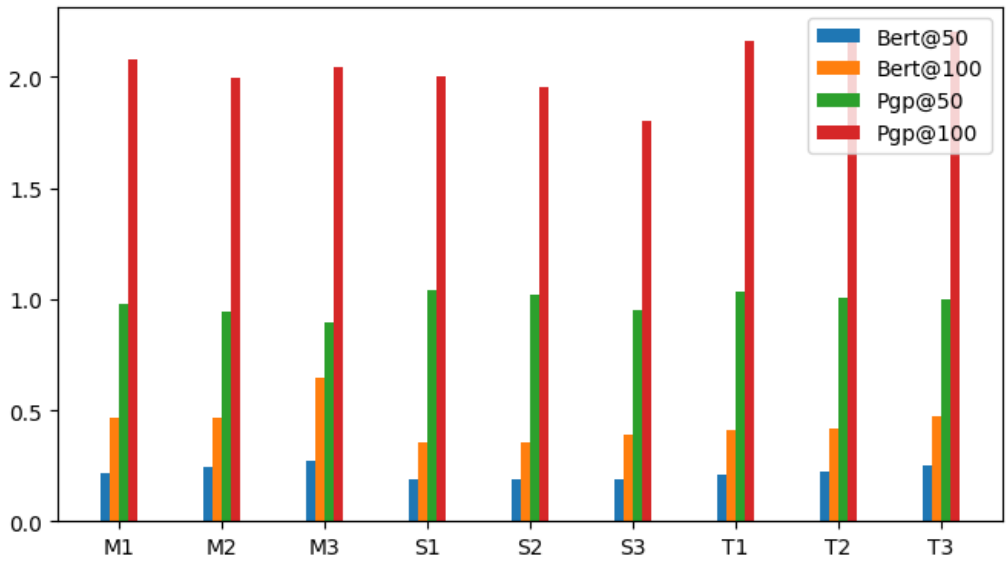

Figura 6: Errori erroneamente introdotti dal processo di correzione per ogni singola frase

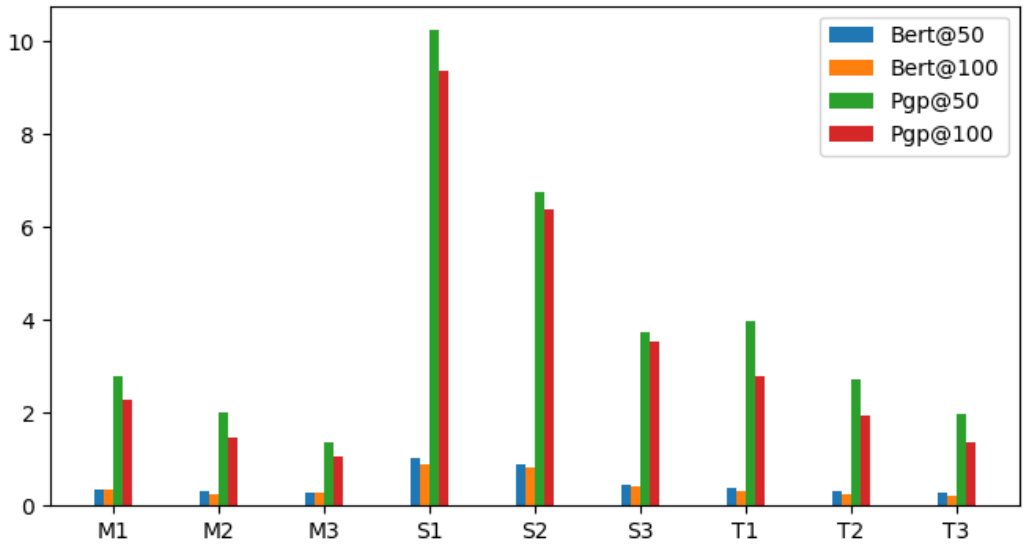

Figura 7: Rapporto fra gli errori erroneamente introdotti dal processo di correzione e gli errori corretti dallo stesso processo

Come previsto in precedenza, si nota un miglioramento in termini di risultati in entrambi i metodi. Ciò sta a significare che il maggior contesto intorno ad ogni errore aiuta effettivamente il processo di correzione.

4.2 Metriche aggiuntive

Data una tupla di tre frasi, ovvero un singolo elemento del dataset, formata come segue:

- Frase originale non perturbata, da qui t_{or};
- Frase sottoposta a perturbazione, da qui t_{pe};
- Frase perturbata alla quale è stato applicato un metodo di correzione, da qui t_{co},

e indicando come segue la distanza di Levenshtein

$$d(x, y) = z \text{ con } x, y \text{ stringhe di testo e } z \in \mathbb{N} \tag{3}$$

posso definire le seguenti metriche:

4. **Riduzione nella distanza di Levenshtein**: informalmente va a misurare quanto la correzione avvicina t_{pe} a t_{or} tramite il processo di correzione risultante nella frase t_{co}. Formalmente è definita come segue:

$$\sum_{t \in D} d(t_{or}, t_{pe}) - d(t_{or}, t_{co}) \tag{4}$$

dove t è una qualsiasi tupla del dataset D. In questa metrica, valori numerici più alti indicano un risultato migliore.

5. **Distanza di Levenshtein totale**: misura quanto le frasi sottoposte a correzione distano dalle frasi non perturbate. Più formalmente:

$$\sum_{t \in D} d(t_{or}, t_{co}) \tag{5}$$

dove t è una qualsiasi tupla del dataset D. In questa metrica, valori numerici più bassi indicano un risultato migliore.

In seguito sono proposti i risultati ottenuti includendo le nuove metriche nei test. Non essendo le due nuove metriche normalizzate sulla lunghezza delle frasi, in questo caso non è possibile comparare direttamente le misurazioni effettuate su dataset con frasi di diversa lunghezza.

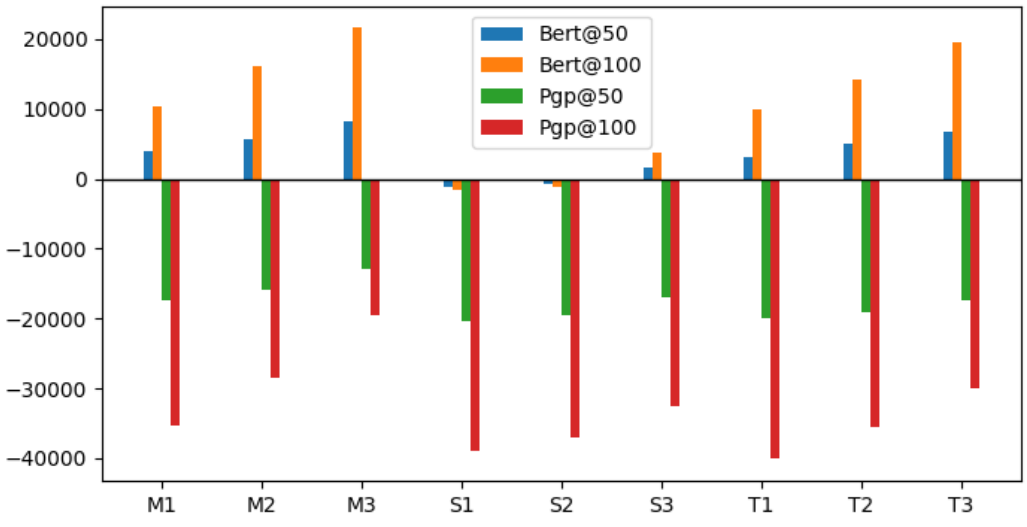

Figura 8: Riduzione nella distanza di Levenshtein

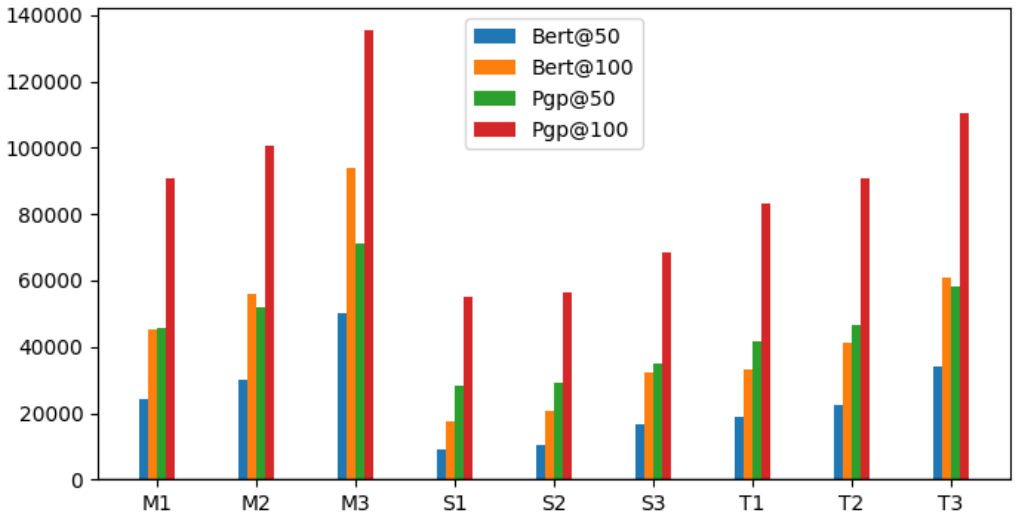

Figura 9: Distanza di Levenshtein totale

I risultati sembrano confermare quanto visto nei precedenti test, senza particolari sorprese.

Riferimenti bibliografici

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Stefan Schweter. Italian bert and electra models, November 2020.

[3] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

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