Misspelling correction using HMM

Andrea Caronni, Riccardo Frigerio, Andrea Guzzo

Sommario

Il presente elaborato presenta la relazione della nostra implementazione del progetto di correzione automatica degli errori ortografici, tramite un Hidden Markov Model

Indice

1 Introduzione

L'obiettivo del progetto è di sviluppare un'applicazione Python la quale, tramite l'utilizzo di un Hidden Markov Model (da qui in poi chiamato HMM), sia in grado di correggere degli errori di battitura su una tastiera elettronica sulla base di alcuni file testuali, contenenti dei tweets reperiti dal popolare social network, i quali vengono utilizzati come dizionari di training.

1.1 Assunzioni

Per questa nostra applicazione abbiamo previsto di correggere un tipo di errore molto comune durante la digitazione su qualsiasi dispositivo dotato di tastiera elettronica ovvero quello che viene chiamato: Fat Fingers. Fat Fingers è un nome che viene attribuito agli errori commessi da parte di un utente che preme una lettera errata sulla tastiera, solitamente vicina, in termine di posizione, a quella che in realtà si vorrebbe premere. Ad esempio al posto di premere la lettera H, l'utente preme la lettera G. Questo errore frequente durante la digitazione provoca un elevato numero di errori soprattutto nei dispositivi mobili. I normali correttori sono in grado di correggere questa tipologia anche a seconda della lingua. Nel nostro caso abbiamo predisposto la nostra catena di Markov per correggere errori Fat Fingers nella lingua Inglese in quanto è stato più semplice reperire dei dizionari di training da twitter e costruire le matrici necessarie per l'addestramento. Lo schema della tastiera a cui abbiamo fatto riferimento è la QWERTY internazionale, anche se il programma è stato costruito per utilizzare diversi schemi di tastiere possibili (aggiungendo file di configurazione in formato json delle tastiere all'interno di una cartella di progetto).

Un HMM inoltre è in grado di correggere non solo parole singole agendo sulle lettere che le compongono, in accordo con i paper allegati alla consegna del progetto, ma anche su coppie di parole e quindi ad un livello di complessità superiore. In particolare è possibile costruire delle matrici di emissione e transizione in base alle coppie di parole che compaiono nel testo. All'inizio del progetto abbiamo tentato questo approccio che si è rivelato fallimentare in quanto la numerosità dei dati, delle matrici e degli stati che componevano la rete erano troppo elevati per le nostre macchine su cui abbiamo svolto questo progetto generando quindi problemi sulla capacità di calcolo e sulla capacità di memorizzazione e di predizione.

Le matrici di emissione e transizione inoltre sono state calcolate a partire dai dizionari raccolti da twitter, è possibile quindi unire diversi dizionari di twitter assieme per generare dei dizionari più corposi incrementando le potenzialità di correzione del modello. Le matrici inoltre sono state accuratamente normalizzate utilizzando i metodi di normalizzazioni del pacchetto scikit-learn. Per la matrice delle emissioni inoltre è possibile calcolarla su un file di testo qualsiasi, non solamente su un dizionario proveniente da twitter.

Per il modello abbiamo implementato due soluzioni: un algoritmo scritto da noi che non abbiamo incluso nella soluzione finale in quanto non aveva buone perfomance e un algoritmo di generazione del modello e di correzione di parole tramite Viterbi utilizzando la libreria Red-Devilz: `http://hidden-markov.readthedocs.io/en/latest/`
Per la pagina di github della libreria: `https://github.com/rahul13ramesh/hmm_project`

1.2 Config.ini

Il file *config.ini* contiene alcune configurazioni utili per il programma ed è posto nella directory contenente il codice dell'applicazione. All'interno di esso, si trovano alcune stringhe utili le quali vengono utilizzate in modo diffuso in più parti dell'applicazione, come i parametri di configurazioni per l'utilizzo delle API di Twitter, o le stringhe che corrispondono ai percorsi di alcuni file cruciali per il funzionamento dell'esecuzione. Il file di configurazione è reperibile, in qualsiasi punto dell'applicazione, tramite le istruzioni:

```
from configparser import ConfigParser
config = ConfigParser()
config.read('config.ini')
```

2 Implementazione del codice

In questa sezione viene descritta la struttura del progetto e l'organizzazione degli script.
All'interno della directory */code* si trovano tutti gli script utili all'esecuzione del progetto, in particolare sono presenti alcune sottodirectory che dividono il codice nei in package, ciascuno per un aspetto particolare dell'applicazione.

2.1 Struttura delle directory

- *tweets_download*: attraverso *tweet_main* permette di effettuare il download dei tweet mediante la libreria Tweepy ed eseguire pulizia del contenuto.
- *generate_model*: si occupa di costruire il modello a partire da un dizionario di riferimento e dal layout della tastiera impostato; costruisce dunque le matrici di transizione ed emissione, per permettere infine la computazione dell'algoritmo di Viterbi.
- *models_IO*: per garantire un'esecuzione più fluida del processo di correzione, il modello, una volta generato, viene salvato ed eventualmente rispristinato dalla memoria attraverso opportuni metodi.
- *words_perturbation*: viene utilizzato per generate il 10% di errori di scrittura come da specifica.
- *evaluate_correction*: contiene gli script utili alle valutazioni della capacità di correzione dell'algoritmo di viterbi, applicato al modello generato.
- *resources*: contiene un file .json che mappa per ogni pulsante della tastiera QWERTY i tasti adiacenti, consente inoltre di importare diversi schemi di mappatura a seconda della tastiera che si vuole utilizzare.

2.2 Entry points

Per eseguire la demo dell'applicazione al fine di inserire delle frasi personalizzati e ottenere, in tempo reale, una correzione dal modello istruito, è sufficiente lanciare lo script *main.py*, il quale lancia l'interfaccia grafica utente attraverso PyQt5. L'interfaccia si presenterà come spiegato nella sezione 6 e potrà essere utilizzata con le modalità riportate sempre nella medesima sezione.
Per eseguire, invece, l'applicazione a scopo di fornire in input un file di testo, eseguire una perturbazione e successiva correzione, e successivamente calcolare le metriche di valutazione del modello, è necessario lanciare lo script *evaluate_test.py*, modificando il codice sorgente e inserendo:

- nella riga 38: il path del file di testo che si vuole utilizzare come input della perturbazione
- nella riga 39: il nome del file di testo che si vuole utilizzare come input della perturbazione

Il file di testo corretto sarà creato automaticamente dall'applicazione in */tweets/corrected/nome_del_file_da_perturbare.txt*.

3 Acquisizione dei tweet

I tweet vengono acquisiti tramite una libreria esterna, reperita in rete, il cui nome è Tweepy (http://www.tweepy.org/). Successivamente, essi vengono processati al fine di rimuovere caratteri speciali che interferirebbero con la corretta computazione dell'HMM. La gestione dell'acquisizione è implementata dallo script *tweet_main.py* contenuto in */code/tweets_download*.
I tweet scaricati, e a disposizione dell'applicazione di test e demo per la computazione del modello, appartengono agli account:

- @realDonaldTrump
- @Forbes
- @MercedesAMG
- @rogerfederer

3.1 Download dei tweet

Il download avviene utilizzando la libreria Tweepy, dopo aver preventivamente creato un'applicazione, tramite l'area sviluppatori di Twitter, allo scopo di avere accesso ad alcune stringhe di configurazione le quali fungono da parametro per interfacciarsi col portale. Nello specifico, le stringhe sono:

- consumer_key
- consumer_secret
- access_token
- access_token_secret

Una volta settato l'accesso per l'API di interfaccia, vengono scaricati i tweets degli account richiesti e memorizzati in in */tweets/raws*.

3.2 Pulizia dei tweet

La pulizia dei tweet è realizzata tramite semplici espressioni regolari le quali, dopo i loro controlli, lasciano intatti solo i puri caratteri di testo e lo spazio. Inoltre abbiamo predisposto una pulizia dei tweet utilizzando la libreria NLTK. Ogni singolo file di tweet scaricato viene esaminato e dopo la sua pulizia viene creata una versione "pulita", la quale viene depositata in */tweets/cleaned*.

4 Perturbazione dei testi

Per introdurre una data percentuale di errore nel testo, e quindi nelle singole parole, è stato implementato un apposito algoritmo di perturbazione, il quale prende in input il file da perturbare, trasformandolo in una stringa, e restituisce in output la stringa perturbata, eventualmente memorizzata in un file.

Algorithm 1 Algoritmo di perturbazione di una percentuale *err* di un testo T

```
1:  procedure PERTURB(T, err)
2:      if T is a file then
3:          T ← T.toString()
4:      words ← number of words in T
5:      words_to_perturb ← ceil(words x err/100)
6:      for i:=1 to words_to_perturb do
7:          word_to_perturb ← T[random(1,words)]
8:          letters ← number of letters in word_to_perturb
9:          letters_to_perturb ← ceil(letters x err/100)
10:         for j:=1 to letters_to_perturb do
11:             letter_to_perturb ← word_to_perturb[random(1,letters)]
12:             new ← neighbour in QWERTY keyboard (no whitespace) of letter_to_perturb
13:             change letter_to_perturb in word_to_perturb with new
        return T eventually as a file
```

4.1 Commento

Per prima cosa viene calcolato il numero di parole da perturbare, la quale corrisponde al tetto della percentuale, di parole che compongono il testo, indicata per il calcolo della perturbazione. Nel caso della consegna, questa corrisponde al 10%. Successivamente, vengono randomicamente individuate le parole, in quantità pari alla percentuale sopra calcolata, sui cui si andrà ad effettuare la perturbazione.
In maniera analoga, si individua il numero di lettere da perturbare per ogni parola, corrispondente alla percentuale, del numero delle lettere totali di quella parola indicata per il calcolo della perturbazione. Nel caso della consegna, questa corrisponde al 10%. Successivamente, vengono randomicamente individuate le lettere, in quantità pari alla percentuale sopra calcolata, sui cui si andrà ad effettuare la perturbazione.
La perturbazione va ad alterare le lettere designate non in maniera casuale, ma sostituendo ognuna con una lettera che le è adiacente sulla tastiera, secondo l'approccio "fat finger", simulando quindi un errore di digitazione. Sono escluse da questo ragionamento, però, le sostituzioni degli spazi, in quanto andrebbero a influenzare pesantemente le metriche di valutazione.

5 Il modello

Il problema è stato modellato, appunto, tramite un Hidden Markov Model il quale viene istruito sulla probabilità di incorrere certi errori di battitura, in certe situazioni di porzioni di parole/frasi già scritte. Successivamente, una volta che il modello viene generato, si utilizza l'algoritmo di Viterbi per eseguire il task di inferenza di calcolo della sequenza più probabile di stati, data una certa serie di osservazioni. Di seguito, vengono specificati i parametri che compongono il modello:

5.1 Parametri del modello

5.1.1 Stati

Gli stati nascosti del modello sono equivalenti a tutte le lettere dell'alfabeto inglese, presenti nella tastiera QWERTY, nella loro forma maiuscola e minuscola, compreso il carattere di spaziatura. Gli stati sono, in termini più pratici, o la riga (o la colonna) che corrisponde all'indice per le righe (o le colonne) della matrice delle transizioni.

5.1.2 Emissioni

Le emissioni sono equivalenti a tutte le lettere dell'alfabeto inglese, presenti nella tastiera QWERTY, nella loro forma maiuscola e minuscola, compreso anche il carattere di spaziatura. Le emissioni sono, però, in termini più pratici, la riga che corrisponde all'indice per le righe della matrice delle emissioni, ovvero tutte le lettere che è possibile digitare correttamente/erroneamente al posto di una lettera associata ad uno stato.

5.1.3 Probabilità a priori

La probabilità a priori corrisponde alla prima riga della matrice delle transizioni, in quanto è la probabilità di incontrare ciascun carattere contemplato nella tastiera QWERTY avendo appena realizzato uno spazio. Nella fattispecie della prima osservazione, questa corrisponde alla probabilità di osservare ciascun carattere contemplato nella tastiera QWERTY non avendo osservato nulla prima di quel momento.

5.1.4 Matrice delle transizioni

La matrice delle transizioni è calcolata sulla base dei dizionari di training, ovvero dei tweet scaricati e puliti tramite espressioni regolari. Questi vengono percorsi da una funzione che calcola la frequenza assoluta di qualunque bigramma nel testo e realizzando successivamente una normalizzazione, ottenendo una matrice compatibile con quella di un HMM. La matrice delle transizioni verrà meglio spiegata nel capitolo 4.2.1.

5.1.5 Matrice delle emissioni

La matrice delle emissioni corrisponde alla probabilità che ciascun carattere contemplato nella tastiera QWERTY sia stato correttamente digitato data la lettera

che in realtà si voleva digitare. In termini pratici, quanto vale la probabilità che si osservi una certa lettera dell'alfabeto, associata all'osservazione, volendo intendere la lettera associata allo stato. La matrice delle emissioni verrà meglio spiegata nel capitolo 4.2.2.

5.2 Calcolo del modello

In maniera preliminare, il software controlla se è già stato calcolato il modello corrispondente ai dizionari scelti come input. Se è già presente un modello preventivamente calcolato, questo viene ricaricato tramite parsing dei file in cui sono stati memorizzati i relativi parametri. Nel caso in cui non esista un "backup" del modello, viene lanciata la sua effettiva computazione.

5.2.1 Calcolo della matrice delle transizioni

I vari dizionari di input, corrispondenti ai tweet puliti, vengono fusi in un unico file testuale, il cui nome è composto dai nomi dei dizionari concatenati l'uno con l'altro; il dizionario unico viene posto in */dictionaries*.
Tale file funge da input per la creazione della matrice delle transizioni: il dizionario viene scansionato e, per ogni carattere di tastiera QWERTY trovato, viene calcolato il numero di occorrenze di bigrammi composti da quel carattere e qualsiasi altro carattere di tastiera QWERTY.
Si ottiene dunque una matrice quadrata, di numero di righe/colonne pari al numero di caratteri incontrati nel dizionario, dove la cella *ij* corrisponde al numero di occorrenze del bigramma <carattere i, carattere j> nel testo.
Tale matrice non è però sufficiente a far funzionare l'HMM, in quanto se il dizionario di input non contiene almeno un carattere che compongono il testo da correggere, il modello non funzionerà interrompendo l'esecuzione. E' dunque necessario completare la matrice aggiungendo tutti quei bigrammi non incontrati, basandosi sulla conoscenza della tastiera QWERTY o di una tastiera corrispondente contenuta ad esempio nel file *qwerty_ simple.json* presente in */code/resources*.
La matrice completa viene poi normalizzata in modo da creare una matrice fruibile come parametro per il modello HMM.

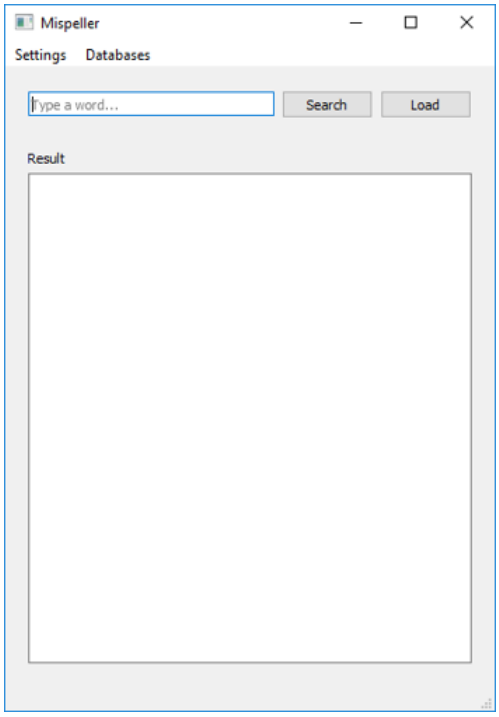
5.2.2 Calcolo della matrice delle emissioni

Sulla base della conoscenza della tastiera QWERTY contenuta nel file *qwerty_ simple.json* presente in */code/resources*, viene creata una matrice delle emissioni seguendo una logica per cui la probabilità di digitare un certo carattere intendendone un altro è distribuita tramite una gaussiana dove:

- Il picco di probabilità è realizzato per la lettera osservata uguale alla lettera che si intendeva digitare;
- La probabilità rimanente è distribuita nelle lettere che si trovano come vicini di quella lettera sulla tastiera QWERTY
- Le restanti probabilità sulla matrice sono configurate a 10^{-5} come probabilità di default.

6 Demo

È stata realizzata un'interfaccia grafica per poter interagire direttamente con l'HMM: per la sua implementazione è stato scelto l'utilizzo di Python sfruttando l'apposita libreria PyQt5.



6.1 Principali funzionalità

- Immissione di testo e sua correzione: una volta digitato il testo, è sufficiente premere il bottone "Search" per avviare la computazione dell'algoritmo di Viterbi per ottenere la correzione del testo digitato;
- Immissione di testo e sua correzione in tempo reale: utilizzando il menu a tendina "Settings", è possibile attivare/disattivare la correzione in tempo reale del testo immesso tramite pressione della voce "Live correction". Se attivata, la correzione live lancia l'algoritmo di Viterbi ad ogni immissione/cancellazione/modifica di un carattere nella TextBox, restituendo la correzione del testo in quel momento inserito;
- Caricamento di file .txt e sua correzione: utilizzando il bottone "Load", è possibile caricare un file di testo in formato .txt dal file system. Al termine del suo caricamento, verrà visualizzata la correzione calcolata tramite algoritmo di Viterbi;
- Selezione dei dizionari di input per la costruzione del modello: per eseguire dei test su dizionari specifici, è possibile selezionare/deselezionare ciascuno di questi tramite il menu a tendina "Databases". Le possibilità rispecchiano i dizionari ricavati dai tweet degli account indicati all'inizio della relazione. Anche in questo caso, ogni volta che un modello costruito sui dizionari settati è già presente in memoria, esso viene ripristinato; in caso contrario viene generato il nuovo modello.

7 Valutazione del modello

Nell'implementazione del codice, come spiegato nella sezione 4.1, esiste una sottodirectory chiamata *evaluate_correction* la quale contiene una serie di script il cui scopo è effettuare il calcolo delle metriche per il modello.

7.1 Implementazione del calcolo delle metriche

7.1.1 Entry point: *evaluate_test*

Il modulo, contenuto in */code*, contiene una serie di fasi che consentono la creazione del modello e la correzione di un file di testo a scelta in input. E' possibile, nelle prime righe dello script, tramite modifica del codice sorgente, selezionare quali dizionari includere per la computazione del modello; successivamente, verrà lanciata la creazione di quest'ultimo, utilizzando il core dell'applicazione.
Una volta che il modello è stato trainato, è necessario impostare path e nome del file di testo che si intende fornire in input per la perturbazione e conseguente correzione, al fine di calcolare le metriche. Le modalità di impostazione sono descritte nella sezione 4.2. Successivamente, il modulo procede alla perturbazione del file fornitogli come input e alla sua correzione, lanciando l'algoritmo di Viterbi tramite il modello preventivamente istruito.
Avendo ora a disposizione il file originale di input, il file perturbato, il file corretto, è possibile valutare le performance del modello confrontandoli. In ultimo luogo, le metriche di valutazione vengono stampate.

7.1.2 *evaluate*

Il contenuto dello script è una serie di metodi per il calcolo delle seguenti metriche di performance:

- Tasso di parole perturbate e corrette
- Tasso di parole non perturbate e non corrette
- Precisione
- Recall
- Accuratezza
- F1 score

Il modulo contiene anche una serie di funzioni di supporto per capire se una parola è stata perturbata, corretta o eventualmente invariata dal testo originale.

7.1.3 *evaluate_utility*

Il modulo include una serie di metodi di supporto per il caricamento dei file corretti/perturbati/originali, per la scrittura su file di alcuni dati temporanei, per la pulizia di testo.

7.2 Valutazione dei risultati

L'idea per valutare le performance del modello è molto semplice: si scorrono i file contenenti il testo perturbato, corretto e originale e, ad ogni parola, viene associata una tripla di interi che indica se tale parola è stata perturbata e/o corretta e/o corrisponde alla verità originale. Sulla base di questa tripla, è possibile poi calcolare se il modello si è comportato più o meno correttamente.
La tripla è dunque del tipo

$$(n, n, n)$$

dove

$$n \in \{0, 1\}$$

e dove ogni intero rappresenta quanto segue:

1. Vale 1 se la parola è stata modificata dal meccanismo di perturbazione, 0 altrimenti.

2. Vale 1 se la parola è stata oggetto di tentativo di correzione da parte del modello, 0 altrimenti.

3. Vale 1 se la parola corrisponde a ciò che è la verità che si intendeva raggiungere, 0 altrimenti.

Rappresentiamo il significato di ogni combinazione dei tre interi tramite la tabella 1. Occorre precisare che alcune combinazioni risultano in una correzione impossibile per il modello; verrà tenuto conto di tali parole durante il calcolo finale delle metriche. Le parole si andranno dunque a dividere in quattro categorie, in base al tentativo

Tripla				
Perturbata	**Corretta**	**Esatta**	**Descrizione**	**Valutazione**
0	0	0	S.V.	?
0	0	1	Giustamente non corretta	✓
0	1	0	Ingiustamente corretta	X
0	1	1	S.V.	?
1	0	0	Ingiustamente non corretta	X
1	0	1	S.V.	?
1	1	0	Correzione errata	X
1	1	1	Correzione esatta	✓

Tabella 1: Descrizione del significato di ogni possibile tripla attribuita ad una parola in output

del modello di correggerle e a quanto questo tentativo è riuscito o meno:

- **giustamente corrette**: parole che sono state perturbate e corrette dal modello in modo da riportarle coincidenti con l'originale. Corrispondono alle parole contrassegnate con la tripla (1, 1, 1);

- **ingiustamente corrette**: parole che sono state corrette dal modello in maniera errata. Corrispondono alle parole contrassegnate con la tripla (1, 1, 0) o (0, 1, 0);

- **giustamente non corrette**: parole non che sono state perturbate e non sono state corrette dal modello, e corrispondono all'originale; Corrispondono alle parole contrassegnate con la tripla (0, 0, 1);
- **ingiustamente non corrette**: parole per cui la necessaria correzione non è stata eseguita. Corrispondono alle parole contrassegnate con la tripla (1, 0, 0).

Posiamo, dunque, rappresentare la matrice di confusione dei dati in output tramite la tabella 2:

		Risultato del modello	
		T	F
Verità nascosta	T	TP: (1,1,1)	FN: (1,0,0)
	F	FP: (0,1,0), (1,1,0)	TN: (0,0,1)

Tabella 2: Matrice di confusione

7.2.1 Metriche calcolate

Dalla matrice di confusione sopra definita, è possibile calcolare le metriche in maniera semplice. Occorre ricordare che la nostra applicazione scorre i 3 file di output e memorizza le triple per ogni parola in un dizionario: in questo modo, diventa facile contare quante triple di una certa combinazione vengono costruite. Infatti, nell'esplicitare le metriche sotto riportate, riportando la tripla (n, n, n) intendiamo la quantità di triple (n, n, n) risultanti nel dizionario finale.

Tasso di parole perturbate e corrette
Metrica che misura il numero di parole di un dizionario che sono state perturbate attraverso l'algoritmo e sono state corrette con successo da Viterbi

$$Tasso\,di\ parole\,perturbate\,e\,corrette = \frac{(1,1,1)}{(1,1,1)+(1,1,0)+(1,0,0)}$$

Tasso di parole non perturbate e non corrette
Metrica che misura il tasso di parole che non sono state perturbate, ma che per qualche motivo non sono corrette secondo Viterbi in base al dizionario

$$Tasso\,di\,parole\,non\,perturbate\,e\,non\,corrette = \frac{(0,0,1)}{(0,0,1)+(0,1,0)}$$

Precisione
Rappresenta la proporzione di tutti i risultati corretti ritornati dal modello

$$Precisione = \frac{(1,1,1)}{(1,1,1)+(1,0,0)}$$

Recall

E' la frazione dei risultati corretti rispetto ai valori significativi ritornati dal modello

$$Recall = \frac{(1,1,1)}{(1,1,1) + (0,1,0) + (1,1,0)}$$

Accuratezza

L'accuratezza misura la bontà del nostro modello tra i vari risultati positivi di correzione ottenuti

$$Accuratezza = \frac{(1,1,1) + (0,0,1)}{(1,1,1) + (1,0,0) + (0,1,0) + (1,1,0) + (0,0,1)}$$

F1 Score

F1 Score è la media pesata rispetto alla precisione e il recall, ha valori da 0 a 1

$$F1 - score = 2 * \frac{precisione * recall}{precisione + recall}$$

7.3 Valutazione delle performance

Per la valutazione delle performance abbiamo utilizzato due file di test: il primo generato estraendo alcune percentuali dai dati ottenuti dal dizionario completo dei tweet raccolti, mentre il secondo lo abbiamo ottenuto dal seguente sito web: `https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html` e contiene una vasta raccolta di conversazioni ottenute dai sottotitoli di diverse commedie in lingua inglese.
Dopo aver ottenuto i file abbiamo valutato le performance utilizzando due modalità:

1. Abbiamo utilizzato il 20%, e successivamente il 30%, del dizionario dei tweet applicando varie percentuali di errori Fat Fingers per poi correggere il file e valutare le performance di correzione rispetto al nostro dizionario utilizzato.

2. Abbiamo utilizzato il file con i sottotitoli applicando il 10% di perturbazione sulle parole e sulle frasi per valutarne le performance di correzione.

Il risultato finale è quindi la comparazione tra i file in input normali, i file perturbati e la loro correzione, utilizzando come valutazione le metriche elencate precedentemente.
Di seguito inseriamo la tabella comparativa dei risultati ottenuti a partire dal dizionario dei tweet.

Dictionary	20%					30%		
String perturbation	10%	20%	40%	50%		10%	30%	40%
Word perturbation	10%	20%	40%	50%		10%	30%	40%
Perturbed corrected ratio	0,3823	0,2742	0,2279	0,2047		0,3823	0,2650	0,2246
Precision	0,5472	0,5308	0,4921	0,4773		0,5591	0,5200	0,4919
Not perturbed not corrected ratio	0,8713	0,8721	0,8724	0,8745		0,8685	0,8712	0,8713
Accuracy	0,8326	0,7340	0,6766	0,6300		0,8306	0,7311	0,6757
Recall	0,1754	0,2317	0,2155	0,2059		0,1696	0,2239	0,2109
F1-measure	0,2657	0,3226	0,2998	0,2877		0,2602	0,3130	0,2952

Di seguito inseriamo la tabella comparativa dei risultati ottenuti a partire dal file dei sottotitoli

Sottotitoli	100%	
String perturbation	10%	>10%
Word perturbation	10%	>10%
Perturbed corrected ratio	0,3422	Too hard to compute
Precision	0,4970	" "
Not perturbed not corrected ratio	0,8542	" "
Accuracy	0,8146	" "
Recall	0,1430	" "
F1 - measure	0,2221	" "

8 Conclusioni e possibili sviluppi

La capacità di correzione del nostro modello diminuisce al crescere dell'input e della percentuale di perturbazione, si comporta bene con parole singole e con un testo attorno alle 60.000 parole in input. Al crescere dell'input inoltre, oltre ad un peggioramento delle metriche di valutazione, abbiamo riscontrato problemi nell'esecuzione del programma in quanto il numero di elementi da analizzare era troppo elevato e le nostre macchine non avevano una capacità di calcolo sufficiente per gestire un input superiore in media alle 100.000 parole. I nostri test sono stati eseguiti utilizzando un dizionario di 31.000 parole ottenuto da Twitter. Al crescere dell'input possiamo quindi evidenziare un peggioramento nella capacità di correzione utilizzando, come dizionario di addestramento, la collezione dei tweets scaricati. I risultati di precision-recall sono buoni: abbiamo dei sufficienti valori di precisione e dei bassi valori di recall in quanto i risultati di correzione rispetto al dizionario sono pochi, ma con un livello di correttezza sufficiente in quanto vengono ritornati ed analizzati rispetto al totale solamente le parole che sono state perturbate e corrette.
Sicuramente utilizzando una numerosità di tweets maggiore, ottenuti tramite un'accurata indagine su un più largo spettro di contesti differenti, potrebbe aumentare notevolmente la capacità correttiva del modello. Inoltre, una più accurata implementazione ed ottimizzazione della soluzione comporterebbe un minore tempo di calcolo e un incremento della capacità correttiva e perturbativa dei file di test. A causa delle problematiche riportate in questa sezione, non abbiamo potuto sperimentare e correggere il file contenente i sottotitoli dei film con una percentuale di perturbazione superiore al 10% per le parole e al 10% del testo,in quanto conteneva una numerosità di parole troppo elevata da correggere e da validare.

8.1 Possibili Sviluppi

Oltre al miglioramento della soluzione, sia in termini di capacità di calcolo, ottimizzazione del codice e velocità di esecuzione, utilizzare altre librerie per HMM potrebbe sicuramente portare ad un aumento delle performance, oltre a fornire un'analisi più accurata rispetto allo stato dell'arte.
Avendo a disposizione delle macchine di calcolo più performanti, si potrebbe riuscire ad ottenere una modellazione del problema non solo dal punto di vista delle singole parole, ma anche dal punto di vista della frase e quindi delle coppie di parole che compaiono nei dizionari. Questa sperimentazione si potrebbe rivelare molto utile incrociando il modello da noi realizzato con questa nuova proposta in modo da fornire una correzione a più livelli, la quale porterebbe ad un livello di correzione più profondo.
Un miglioramento si potrebbe avere rendendo la capacità di correzione a blocchi, in modo da correggere per parole segmenti di testo per poi assemblare l'output finale. In questo modo si riuscirebbe a suddividere il calcolo per renderlo parallelo e conseguentemente più ottimizzato.