Misspelling

Giulia Boifava, 781250
Alessandro Bregoli, 780711
Pietro Brenna, 781840

6 Luglio 2017

Introduzione

Gli odierni strumenti tecnologici quali smartphone, tablet e computer sono dotati di una serie di programmi e software che controllano l'ortografia delle parole digitate, correggendo automaticamente il termine scorretto o proponendo, in fase di digitazione, una serie di dizioni che più somigliano alle potenziali parole a cui si può riferire la sequenza di lettere già battute a tastiera.
Il progetto da noi scelto (Misspelling) ha riproposto questo genere di problema, pertanto abbiamo scelto di provare a implementare anche noi un correttore automatico di parole.

Assunzioni sul dataset

Per semplicità grammaticale e lessicale abbiamo scelto la lingua inglese in quanto scarsa o priva di determinati simboli (accenti) e generi (maschile / femminile). Inoltre abbiamo effettuato un'ulteriore scrematura non considerando simboli di punteggiatura e caratteri speciali quali "#", "-", "@" e simili, ma considerando solamente le lettere.
La piccola modifica che abbiamo effettuato rispetto alla specifica, è il fatto di aver preso in considerazione come dataset un libro di narrativa in lingua inglese, invece che i Tweet del famoso Social Network.

Dataset

Il dataset che abbiamo scelto è composto da testi di romanzi quali:

- Il signore degli Anelli
- Harry Potter
- Dune

La ragione di questa scelta è data dal fatto che il nostro modello prende in considerazione la probabilità di una parola data la precedente.

Scelte implementative

Abbiamo sviluppato il progetto in linguaggio Python 3.6 e abbiamo aggiunto un ulteriore metodo di digitazione: oltre alla normale dicitura ed errori che possono incorrere digitando su una tastiera qwerty, abbiamo anche preso in considerazione i problemi che possono sorgere utilizzando una tastiera Braille.
L'interfaccia è stata sviluppata in HTML. Una volta digitata la parola appare un menù con una lista di 6 alternative, scelte dal programma in base all'adiacenza di eventuali lettere scritte per errore.

Modello

- Per la tastiera qwerty abbiamo considerato adiacenti le lettere vicine. Per esempio, le lettere adiacenti alla “a” sono “q”, “w”, “s”, “z”.
- Per la tastiera Braille, invece, abbiamo considerato adiacenti le lettere che differiscono l’una dall’altra per la presenza o meno di un puntino. Per esempio, abbiamo considerato adiacenti alla “d” () i caratteri “g” (), “n” ( ), “c” ( ), “e” ( ).

Metodo

Operiamo con un hidden markov model in cui:

- gli stati corrispondono alle parole intere
- l'osservazione corrisponde alle lettere digitate per formare la parola
- la transizione è data dalla probabilità di una parola data la precedente, calcolata sul dataset.

In più viene costruito a partire dal dataset un insieme di parole corrette ("dizionario").

Modello di osservazione

Come calcolare la probabilità di uno stato (parola) data l'evidenza (lettere digitate)?
Partiamo dall'intuizione di fondo della edit distance e creiamo una funzione di edit probability, che dà la probabilità di aver digitato una certa evidenza avendo inteso digitare una certa parola. Tale funzione si appoggia ad un modello di digitazione (tastiera o braille nel nostro caso).

Edit Probability

	ϵ	c	i	a	l
ϵ	1.0	0.05	0.0025	0.000125	6.25e-06
c	0.02	0.92	0.0184	0.000368	7.36e-06
i	0.0004	0.046	0.8464	0.016928	0.00033856
a	8e-06	0.0023	0.04232	0.778688	0.01557376
o	1.6e-07	0.000115	0.002116	0.0389344	0.00648906

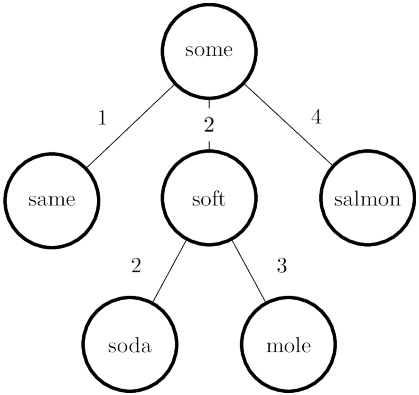
Table 1: Calcolo della edit probability di "ciao" dato "cial"

BK-Tree

Tuttavia la edit probability risulta essere lenta e dunque è stata implementata la struttura dati BK-Tree che risulta essere più efficiente nel selezionare parole sintatticamente vicine a quella digitata.

BK-Tree

Tuttavia la edit probability risulta essere lenta e dunque è stata implementata la struttura dati BK-Tree che risulta essere più efficiente nel selezionare parole sintatticamente vicine a quella digitata.



Correzione errori

Una volta costruito tale modello, l'algoritmo di Viterbi ci restituisce la sequenza più probabile di parole. È importante notare che tutte le parole restituite dall'algoritmo di Viterbi provengono dal dizionario.

Risultati

Per validare il progetto abbiamo preso un testo e abbiamo alterato circa una lettera ogni 30 per avere mediamente una parola errata ogni 7-8. Misuriamo la percentuale di parole corrette prima e dopo aver applicato l'algoritmo di correzione.

Testo	**Perturbazione**	**Errore residuo**
test1	15%	2%
test2	12%	4%

Test 1

	non corrette	corrette
non perturbate	350	0
perturbate	10	54

- Accuracy : 98%
- Sensitivity: 100%
- Specificity: 97%
- Precision : 84%
- NPV : 100%

Test 2

	non corrette	corrette
non perturbate	172	0
perturbate	7	17

- Accuracy : 96%
- Sensitivity: 100%
- Specificity: 96%
- Precision : 70%
- NPV : 100%

Conclusioni

I maggiori problemi dipendono dal fatto che l'HMM è del primo ordine dunque ogni parola dipende solo dalla propria distanza rispetto a quella digitata e dalla parola precedente; inoltre non tiene in considerazione la semantica della frase. In generale il modello corregge con maggiore accuratezza frasi brevi.

Possibili miglioramenti

Possibili sviluppi futuri potrebbero essere:

- Utilizzo di un modello HMM di ordine superiori al primo
- Considerazione della logica della frase utilizzando NER
- Parallelizzazione di alcuni task in modo da rendere il modello più scalabile