Misspelling

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

6 Luglio 2017

Indice

1 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.

Si richiede di basare il lavoro su un Hidden Markov Model; non è immediato però decidere a che cosa corrispondano gli stati: se da un lato è semplice creare un modello in cui le singole lettere sono evidenza e stato, d'altro canto ci sembra ottimistico pensare che la probabilità di una lettera data la precedente sia una misura su cui basare un correttore che vorremmo essere robusto.

Pertanto facciamo la scelta di considerare stati le parole; proponiamo inoltre un metodo per calcolare la probabilità di una parola data l'osservazione di una sequenza di lettere.

2 Dataset

2.1 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.

2.2 Contenuto del dataset

Come dataset abbiamo scelto di utilizzare una serie di romanzi in lingua inglese; i principali sono:

- 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; di conseguenza un libro è ottimo per apprendere queste probabilità a priori; inoltre un libro contiene una grande quantità di parole distinte; ciò permette di creare un corposo dizionario di termini e di analizzare nel dettaglio le relazioni che intercorrono andando a formare una base di conoscenza sufficientemente solida.

3 Scelte implementative

3.1 Strumenti software

Abbiamo sviluppato il progetto in linguaggio Python per via della sua versatilità e della sua popolarità sia nell'ambito scientifico che in quello professionale; inoltre abbiamo mantenuto separata la logica del progetto (ossia il correttore) dall'interfaccia utente che dunque può essere facilmente sostituita. L'interfaccia proposta è un'applicazione web scritta utilizzando HTML CSS e Javascript.

3.2 Metodi di digitazione

Come metodi di digitazione abbiamo considerato due principali strumenti:

- Tastiera QWERTY
- Tastiera Braille

I metodi di input nonostante condividano lo stesso metodo per la correzione di errori si basano su due modelli di errore distinti correlati alla disposizione dei tasti e al loro utilizzo.

3.3 Modello di errore QWERTY

Per la tastiera QWERTY (layout italiano) abbiamo sviluppato una distribuzione di probabilità data la digitazione di una lettera come segue:

- Carattere corretto: 92%
- Carattere errato adiacente: 5%
- Carattere errato non adiacente: 3%

Questi valori sono frutto di una sperimentazione empirica.
Per carattere adiacente si intende un qualunque carattere fisicamente adiacente a quello inserito sulla tastiera; la differenza di probabilità tra caratteri adiacenti e caratteri non adiacenti dipende dal fatto che si suppone una maggiore probabilità di sbagliare una parola con un carattere adiacente a quello che si voleva inserire.

3.4 Modello di errore Braille

Per la tastiera Braille a 8 punti abbiamo utilizzato la stessa distribuzione di probabilità utilizzata per la tastiera QWERTY tuttavia il concetto di adiacenza è differente: le lettere che differiscono l'una dall'altra per la presenza o meno di un puntino sono considerate adiacenti.

Per esempio, abbiamo considerato adiacenti alla "d" () i caratteri "g" ( ), "n" ( ), "c" ( ), "e" ( )

3.5 Struttura della base di conoscenza

La base di conoscenza contiene due strutture dati:

- Un dizionario contenente tutte le parole apprese dai libri nella fase di training
- Una matrice contenente per ogni parola la probabilità di essere seguita da una qualsiasi altra parola

3.5.1 Dizionario

Il dizionario generato a partire dalla fase di training contiene più di 100000 termini; per il nostro algoritmo è fondamentale poter accedere frequentemente ed efficientemente a questo dizionario; inoltre è necessario poter cercare termini sintatticamente simili ma non uguali in modo da poter ottenere probabili correzioni di una parola. Per ottenere tutto ciò è stato implementato un BK-Tree[3] che risulta essere estremamente efficiente.

3.5.2 Matrice di transizione

Questa matrice, data l'enorme dimensione del dizionario, è stata implementata come lista di liste; in questo modo si risparmia una grande quantità di spazio e si vanno a salvare solo le transizioni viste nel dataset. In fase di accesso se si richiede una transizione non presente nella "matrice" il sistema ritorna una probabilità ε molto piccola in modo da non avere mai valori pari a 0 durante l'esecuzione di Viterbi.

3.6 Modello di osservazione

La edit distance[2] è uno strumento molto potente per capire la distanza di un termine da un altro e per questo è stata utilizzata come metrica dal BK-Tree; questa distanza non è però sufficiente; tuttavia partendo dall'intuizione di fondo abbiamo sviluppato 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 al modello di digitazione. L'idea è cercare di massimizzare tale probabilità facendo confronti con ogni parola nel dizionario; per ragioni di performance, effettuiamo prima una scrematura andando a selezionare attraverso il bk-tree solo termini ad una distanza ragionevole rispetto all'evidenza.

3.7 Metodo di correzione

La correzione di frasi si basa su un Hidden Markov Model così definito:

- gli stati corrispondono alle parole intere

	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

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

	ϵ	c	i	a	w
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	0.038934	0.000778688

Tabella 2: Calcolo della edit probability di "ciao" dato "ciaw": inferiore alla precedente perché w è lontano da o sulla tastiera.

- 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.

Ogni parola ha una probabilità di osservazione determinata dalla funzione di edit probability. L'utilizzo congiunto del modello di osservazione e dell'HMM permettono una correzione dipendente dal contesto; tuttavia tale contesto è piuttosto limitato in quanto si parla di un HMM del primo ordine.

3.7.1 Suggerimenti

Una volta che Viterbi viene eseguito la sequenza più probabile viene calcolata e rappresenta la correzione totale della frase da parte del metodo; tuttavia il metodo espone anche dei suggerimenti per l'ultima parola andando ad ordinarli dalla più probabile alla meno probabile.

4 Risultati

Per testare il nostro modello abbiamo utilizzato due racconti i quali sono stati perturbati con errore dipendente dal modello precedentemente presentato:

Testo	Perturbazione	Errore residuo
test1	16%	3%
test2	12%	5%

Tabella 3: Risultati

4.1 Test online sul modello Braille

L'interfaccia Braille riesce a correggere le parole in modo soddisfacente. Prima considera gli errori derivanti dall'aver digitato una lettera in più o in meno, e successivamente verifica se vi è una lettera errata e la corregge con quella più vicina. Esempio: I suggerimenti che appaiono digitando "plint" sono "pliant", "plaint", "pint", "splint" e infine "print" e "point". Risultato accettabile, in quanto le lettere adiacenti alla "l" sono "r", "p", "v" e "b". Ora, non essendoci possibili termini nel dizionario per i quali la "l" si possa sostituire con gli ultimi tre suggerimenti, la potenziale parola è "print". A questo punto, per l'ulteriore proposta, viene calcolata la lettera successiva con la più breve distanza di Levenshtein dalla "l" che generi una dizione esistente. Questa è "o" e di conseguenza il termine suggerito è "point".

4.2 Test online sul modello QWERTY

L'interfaccia HTML per la tastiera QWERTY lavora analogamente alla tastiera Braille. Esempio: Digitando la parola scorretta "bubbke", il correttore colloca al primo posto della lista dei suggerimenti il termine corretto "bubble"; risultato ottimo poiché "k" e "l" sono adiacenti sulla tastiera. Quindi viene suggerita la stessa parola al plurale "bubbles" e a seguire "hubble", dove la "h" è ancora fisicamente vicina alla "b". Non essendoci poi altre potenziali lettere adiacenti alla "b", l'attenzione della correzione si sposta sul carattere successivo; il quarto suggerimento è infatti "bibble". Si noti, ancora una volta, che la "i" è adiacente alla "u". Continuando a scorrere poi la sequenza di lettere della parola sbagliata, e non trovando potenziali correzioni nel dizionario, il correttore ritorna a calcolare la distanza di Edit sul primo carattere. Ora, la lettera più vicina a questa risulta essere "r", che genera il suggerimento "rubble". L'ultima correzione, infine, cambia la "e" con la "y" ("bubbly").

5 Conclusioni e sviluppi futuri

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 sviluppi futuri potrebbero essere:

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

Riferimenti bibliografici

[1] Tarniceriu, A., Rimoldi, B., & Dillenbourg, P. (2015, July). *HMM-based error correction mechanism for five-key chording keyboards.* In Signals, Circuits and Systems (ISSCS), 2015 International Symposium on (pp. 1-4). IEEE.

[2] URL: https://pypi.python.org/pypi/python-Levenshtein/0.12.0 [visitato: 3/7/2017]

[3] Burkhard, W. A., & Keller, R. M. (1973). *Some approaches to best-match file searching.* Communications of the ACM, 16(4), 230-236.