Modelli Probabilistici per le Decisioni

Fabbris Elia 793240, Fumagalli Matteo 793670

Settembre 2018

1 Introduzione

I correttori ortografici moderni scansionano un testo parola per parola ricercando ognuna di esse in un dizionario interno. Nel caso una parola non dovesse venire trovata verrebbe evidenziata e il programma fornirebbe un elenco delle parole più papabili per effettuare la correzione. Correttori più avanzati sono inoltre capaci di riconoscere parole che sono errate nel contesto nel quale si trovano, cercando di comprendere la logica di un testo. Nonostante tutto anche i correttori più moderni e avanzati non hanno precisione assoluta.
Con questo progetto viene proposto un sistema di correzione basato su un modello di Markov nascosto (HMM). Un HMM è modello probabilistico dove la probabilità di osservare un suo stato k all'istante t è determinato solamente dallo stato del modello all'istante *t-1*. Data un certa sequenza X di stati il modello dovrà fornire la probabilità che partendo dallo stato iniziale restituisca X come sequenza, probabilità corrispondente alla massima verosimiglianza della sequenza X.
La seguente relazione è organizzata nel modo seguente. Il capitolo 2 riprende il concetto di modello Markoviano nascosto e di come è stato implementato nel progetto. Inoltre viene descritta la creazione del dataset, il funzionamento del programma e l'interfaccia che è stata sviluppata per dare una dimostrazione del funzionamento. Il capitolo 3 contiene un'analisi sui dati ottenuti, concentrandosi sul cambiamento dato dalla grandezza dei dati in input. Infine nel capitolo 4 vengono mostrate le conclusioni.

2 Hidden Markov Model

2.1 Creazione dataset

Nella prima fase del progetto sono stati raccolti i tweet.
È stato creato uno script python nel quale è stata importata la libreria *tweepy*, necessaria per operare attraverso apposite funzioni sui contenuti presenti nell'applicazione Twitter. In particolare è stata usata la funzione *api.user_timeline(@User,count)*, la quale permette di ottenere un certo numero di tweet, definito in *count*, da un profilo, determinato in *@User*.
Per il progetto sono stati utilizzati account twitter di informazione come CNN, TIME, UsaToday e simili in modo da ottenere dei contenuti abbastanza affidabili sotto il punto di vista della sintassi e della grammatica. Una volta ottenuti i dati, sono stati puliti dai vari link e dalle emoticon che contenevano.
I tweet puliti sono stati così copiati in un secondo file e modificati. La perturbazione dei tweet è stata del 10%, modificando il carattere con i caratteri limitrofi presenti sulla tastiera.
Il passo successivo è stato dividere entrambi i testi in training e in test, precisamente 80% training e 20% test.

2.2 Creazione HMM

Le scelte riguardanti la progettazione sono ricadute su un tipo di modello che riconoscesse i caratteri dell'alfabeto e gli spazi, di conseguenza sono state effettuate delle operazioni prima della creazione del modello.

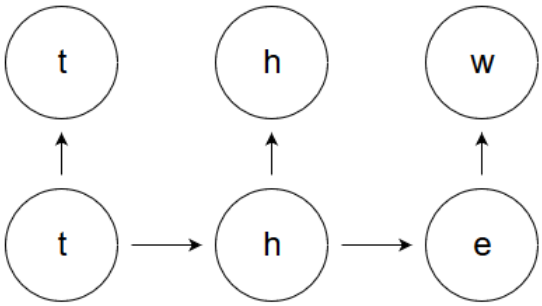
Caratteri accettati			
a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p
q	r	s	t
u	v	w	x
y	z	Space	

Come prima cosa sono state contate le occorrenze di ogni carattere, calcolandone così la percentuale di presenza nel testo di training non modificato.
Successivamente è stata creata la matrice delle transizioni, segnando opportunamente in quest'ultima il passaggio da un carattere precedente al carattere preso in considerazione, per ogni carattere del testo non modificato. La matrice è stata salvata in un file.csv rendendo più facile la consultazione. Gli indici della matrice, sommandoli a 97, corrispondono al carattere seguendo la tabella ASCII; per esempio, la colonna di indice 0 corrisponde al carattere 'a'.
La seconda matrice creata riguarda le osservazioni. In questa fase sono stati utilizzati entrambi i testi di training.
Per ogni carattere presente nel testo di training non modificato è stato controllato il corrispondente nel testo modificato, annotando così l'osservazione e

verificando se quest'ultima corrispondeva a ciò che ci si aspettava. Per necessità è stato scelto un epsilon molto piccolo, pari a 10^{-6}, ed è stato sostituito agli zeri presenti nella matrice. Anche la seconda matrice è stata salvata in un file.csv per un'eventuale consultazione più facile.
Completata questa fase, è stato creato il modello HMM. Come libreria per la creazione di quest'ultimo è stata utilizzata *Pomegranate*. Nel processo di creazione sono state utilizzate alcune funzioni necessarie: inizialmente è stato definito il modello tramite la funzione *HiddenMarkovModel()*, successivamente sono stati creati gli stati tramite il comando *State()* il quale per ogni carattere discretizza la distribuzione di probabilità.
Una volta creati gli stati, sono stati aggiunti al modello e successivamente sono state inserite le transizioni iniziali e quelle tra stati. Per le transizioni iniziali è stata utilizzata la probabilità di occorrenza di ogni stato, invece per le transizioni tra stati è stata utilizzata la matrice creata in precedenza. Con il comando *bake()*, necessariamente prima di qualsiasi calcolo, vengono definiti gli ultimi parametri.
La parte di creazione del modello è terminata, in questo modo è possibile passare alla fase di inferenza, utilizzando i dataset creati.

2.3 Inferenza

Una volta completata la creazione dei dataset e del modello si passa al procedimento di inferenza sui dati, ovvero quella procedura che partendo dalle caratteristiche osservate da una popolazione di train cerca di dedurre quelle di una popolazione di test.
Nella prima fase il file di test contenente i tweet "sporcati" è stato ulteriormente pulito dalla punteggiatura e da vari simboli non riconosciuti dal modello, in modo da ottenere un file contenente solamente caratteri alfabetici minuscoli.
La predizione è affidata all'algoritmo di Viterbi il quale ricava la sequenza di stati più probabili che ha generato la sequenza data in input. Di seguito viene riportato un esempio di come il modello agisce sulla sequenza "thw" come input.



Il programma nel nostro caso processa ciascuna lettera contenuta nella frase, incluse quelle corrette, e salva la sequenza di stati più probabile in una variabile chiamata *"path"*. Dopodichè un ciclo *for* ha il compito di tradurre il nome degli stati contenuti nella variabile in caratteri, concatenandoli uno ad uno ed inserendoli in una stringa. Al termine la frase costruita verrà salvata in un secondo file contenente le correzioni e verrà processata la frase seguente.

2.4 Interfaccia

Una ulteriore funzione che offre il programma riguarda la correzione di una singola frase a discrezione dell'utente, oltre che quindi alla correzione dell'intero file di test.
Il software si interfaccia con l'utente attraverso una finestra di dialogo creata attraverso Tkinter, una libreria che permette la creazione di semplici interfacce grafiche nel linguaggio Python.
Inizialmente viene ricreato il modello utilizzando i dati salvati dalle operazioni di creazione precedente, dopodichè viene eseguito l'algoritmo di Viterbi sulla stringa data in input e, attraverso il procedimento descritto nella sezione antecedente, mostrato il risultato.
Nell'immagine che segue viene mostrato un esempio di esecuzione.

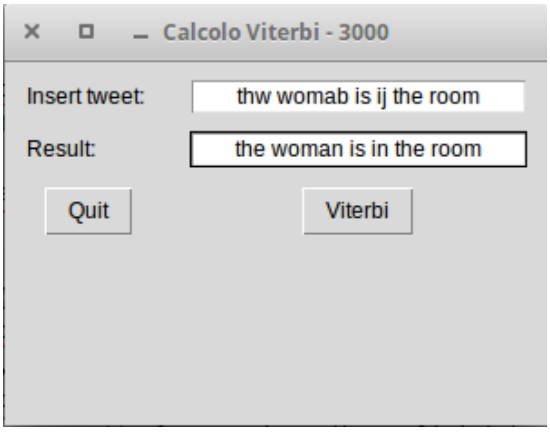


Figure 1: Demo con file 3000 tweet

Figure 2: Demo con file 4500 tweet

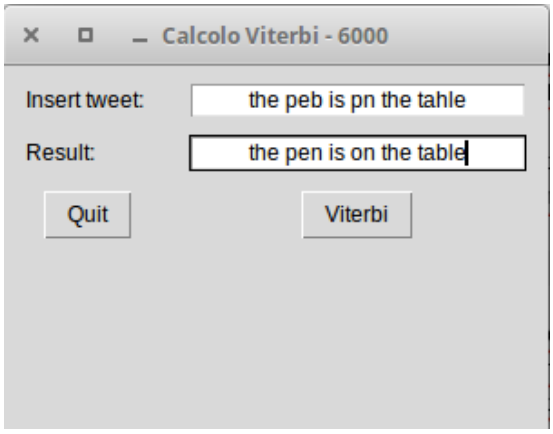
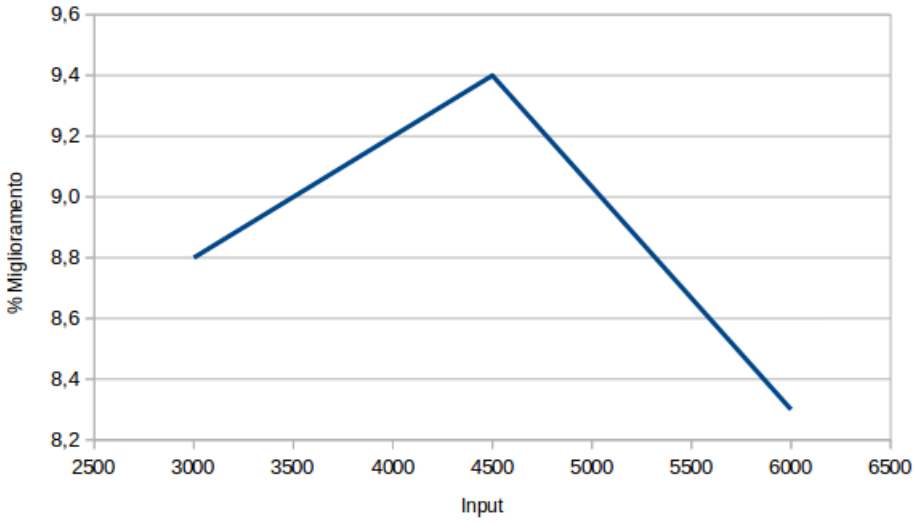


Figure 3: Demo con file 6000 tweet

3 Analisi dei dati

Per la parte di analisi dei dati sono stati usati tre dataset di diversa grandezza, in modo da poter valutare la differenza delle capacità predittive al variare dei dati di training. Precisamente sono stati creati tre dataset composti da 3000, 4500, 6000 tweet. Ognuno di questi dataset è stato diviso in un file di training e in un file di test.
Dopo aver calcolato le sequenze più probabili tramite l'algoritmo di Viterbi ed aver inserito le frasi nel nuovo file, è stato creato uno script python il quale calcola la percentuale di parole corrette sia nel file contenente i tweet "sporchi" sia nel file contenente le frasi corrette tramite l'algoritmo. In questo modo è possibile verificare se è presente un miglioramento o un peggioramento delle prestazioni. Come si può vedere nel grafico sottostante, in tutti i dataset testati c'è stato un miglioramento riguardante la percentuale di parole corrette.



In particolare si può notare che la miglior percentuale di correzione è stata trovata con l'utilizzo del file contentente 4500 tweet. Il motivo per cui la percentuale di miglioramento non cresce al crescere dei dati è che aumentando la quantità di tweet, l'Hidden Markov Model è più sensibile alla percentuale di errori che viene introdotta.
Nella tabella sottostante vengono invece riportati i margini di miglioramento ottenuti eseguendo il programma sui file di test.

Correzione parole			
	Parole totali	Parole uguali - File corretto	Parole uguali - File non corretto
3000 tweet	7919	5506	4809
4500 tweet	11103	7907	6863
6000 tweet	16790	11620	10225

4 Conclusioni

In conclusione è possibile affermare che l'obiettivo del progetto è stato raggiunto, ovvero realizzare un modello che sulla base di un testo scritto ricostruisse la sequenza di stati nascosti più plausibile, corrispondente al testo corretto.
Esistono dei possibili miglioramenti che possono essere apportati al progetto, come per esempio rendere il modello in grado di capire quando è presente una lettera mancante o una lettera di troppo in una parola.
In sintesi tale programma è quindi in grado, sfruttando le potenzialità dei modelli di Markov nascosti, di correggere errori contenuti in un qualsiasi testo dato da un utente.