



Sentiment Analysis su Amazon Reviews

Analisi con algoritmo JST

Danilo Deponi - 737838 Mattia - Curatitoli



Introduzione

Le recensioni online sono diventate un'importante fonte di informazioni sia per i produttori che per i consumatori. Queste recensioni forniscono informazioni su prodotti, brand e utenti. L'analisi di questa quantità crescente di informazioni si concentra sull'identificazione automatica di opinioni, emozioni e giudizi, così come la loro polarità (positiva o negativa). Il risultato finale di questo progetto è stata una *web app* che consente di creare un modello per analizzare l'andamento delle preferenze di un utente nel tempo. In poche parole, si può analizzare il mutamento degli argomenti di un utente in base alle recensioni fatte. Inoltre, è possibile capire se l'argomento trattato ha una tendenza positiva o negativa.

Dataset

Il dataset fornito inizialmente(*.tsv*) contiene una serie di recensioni su *AmazonFood*. Ogni riga di questo dataset mostra: *user_id*, *product_id*, *score* e *review*. Dopo un'accurata analisi del dataset, è stato deciso di scartare questo dataset per molteplici motivi:

- eccessiva quantità di recensioni positive: più dell' 85% delle recensioni ha uno *score* di 4 stelle o superiore
- utenti con pochissime recensioni: la maggior parte degli utenti presenta al massimo una recensione. Solo i primi 10 utenti su 4000 ha più di 10 recensioni (max 30)
- mancanza di un valore temporale nelle recensioni: l'obiettivo posto era quello di analizzare l'andamento dei commenti dell'utente nel tempo, la mancanza di quest'ultimo non permette di effettuare analisi.


Cercando online dataset più corposi ed omogenei, abbiamo trovato un dataset sui *gourmet foods* scaricabile a questo link (*.json*)



Da dataset a SQLite

Per rendere più semplice la gestione delle entità contenute nel dataset, è stato deciso di modellizzare il dataset utilizzando un database *SQLite*. Le principali tabelle ricavate dal dataset sono:

- *user* (id, num_rating, average_score, variance, experience, pos_words, neg_words)
- *product* (id, num_rating, average_score, variance, experience, words)
- *rating* (id, productid, userid, score, text, timestamp).

Più avanti, si avrà modo di capire il significato di alcuni attributi ins  La scelta di modellizzare i dati è stata anche dettata dall' utilizzo del *CMS Django*, utilizzato per la gestione dei contenuti, visualizzazione dei dati e realizzazione della web app.

JST e Topics

Per poter ricavare i topic principali, così come la polarità per ognuno di essi, è stato deciso di utilizzare un modello probabilistico chiamato *JST*(joint sentiment/topic) *model*. *JST* si basa su un più famoso modello detto *LDA* (Latent Dirichlet Allocation). Le 2 peculiarità di *JST* sono la capacità di calcolare simultaneamente sia i topics (ciò che fa *LDA*) che i *sentiment* associati ad essi ed il fatto che sia completamente non supervisionato. *LDA* è essenzialmente un modello a 3 *layer*, dove i topic sono associati ai documenti(nel nostro caso, le *reviews*) e le parole sono associate ai topic. *JST* aggiunge un quarto layer: il *sentiment*; in questo caso il *sentiment* è associato al documento, sotto il quale ogni topic è associato ad un *sentiment*. Le parole, invece, sono associate sia al topic che al *sentiment*.(Fig.1) L' algoritmo *JST*, in questo progetto, è stato

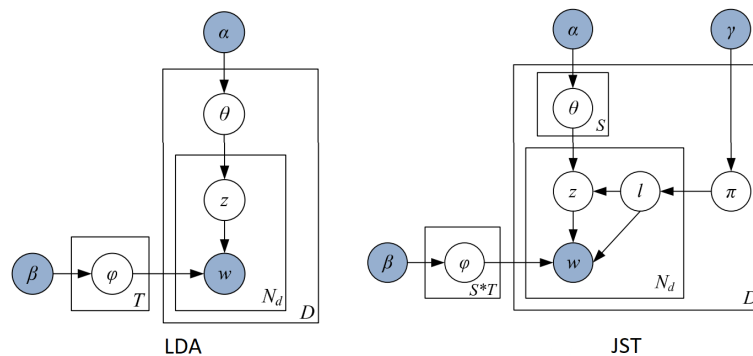


Figura 1:

totalmente implementato in *Python*. L' algoritmo da la possibilità di parametrizzare *numero topic* da trovare, numero di parole per topic e numero di iterazioni (maggiore è

il numero, migliore è la precisione di assegnazione *topic/sentiment*). I topic creati sono salvati nella tabella:

- *topic* (id, name, sentiment, words) .

Analisi dei dati

Una volta generati i topic, oltre alle statistiche (molto semplici) fatte sulle recensioni, viene calcolata la distribuzione dei topic nel tempo per ogni utente. L'idea principale è di intersecare, per ogni topic, le parole positive e negative che caratterizzano l'utente insieme alle parole che caratterizzano il prodotto. Il risultato di questa computazione è una lista derivata dall'intersezione delle parole del prodotto p , con le parole dell'utente u e le parole del topic t . Se viene utilizzato questo metodo su ogni topic e su ogni commento dell'utente (preso in ordine temporale) e poi viene calcolata la lunghezza dell'intersezione trovata, si può avere un indice che indica quanto un utente ha parlato del topic t in quell'istante temporale. La struttura dell'algoritmo è all'incirca la seguente:

```
wt = topicsWords
for u in user:
    wu = userWords(u)
    for t in topic:
        for r in reviews(u):
            p = getProduct(r)
            wp = productWords(p)
            value = len(intersect(wp, wu, wt(t)))
```

Ma come trovare le parole che caratterizzano utente e prodotto? Per fare questo, è stato usato il già citato modello *LDA*. Per ogni recensione positiva (score > 3) di un utente e per ogni recensione negativa (score <= 3) è stata computato *LDA* impostando il numero di topic in base al numero di prodotti recensiti. Sono state poi prese le prime n parole di ogni topic per caratterizzare le preferenze di un utente. Per il prodotto è emerso che la metà del numero di recensioni avute è un buon parametro per impostare il numero di topic di *LDA*. Un altro interessante calcolo è quello sull'esperienza utente: è stato deciso di dare un'indicazione sull'esperienza utente in base alle recensioni fatte. Il calcolo si basa su queste assunzioni:

- l'esperienza dell'utente non è strettamente legata ad un intervallo temporale
- l'esperienza dell'utente matura più effettua recensioni
- utenti esperti tendono a dare opinioni conformi ad altri utenti (soprattutto anch'essi esperti)

Nel nostro calcolo, quindi, è stato assegnato un punteggio di 1 per ogni commento fatto da un utente il più vicino possibile alla massa. Il punteggio diminuisce man mano che

la recensione dell' utente si discosta dallo *score* che ha ottenuto il maggior numero di recensioni.

Visualizzazione dei dati e app

La web app creata permette di scegliere il dataset da importare (purchè sia in un formato supportato) e di popolare le tabelle *SQL* con i dati presenti in esso. Già preliminarmente, è possibile visualizzare dati riguardanti il dataset. Come distribuzione dei commenti, *score medio* per utente ecc..

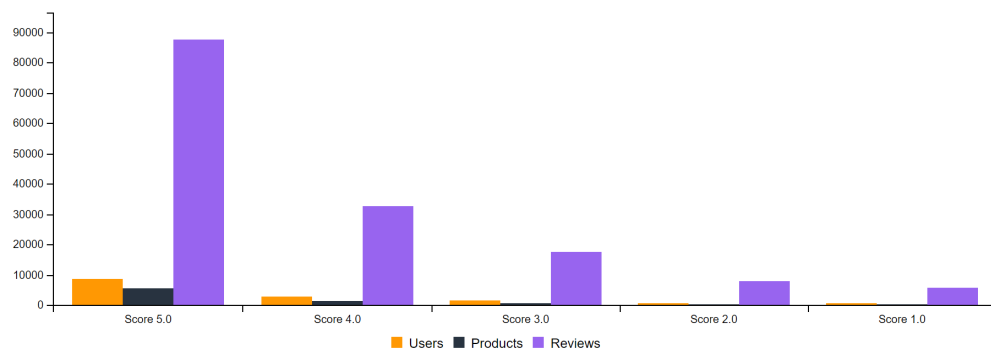


Figura 2: Data per score

E' inoltre possibile visualizzare l'elenco degli utenti e l'elenco dei prodotti, nonchè le *reviews* effettuate. Nella griglia degli *user*, è possibile selezionare gli utenti per i quali si desidera creare il modello, oltre che impostare i topic desiderati, parole per topic ed iterazioni.

Amazon reviews analytics					DATASET DASHBOARD USERS PRODUCTS REVIEWS TOPICS				
N° words		N° topics	N° iterations		Make model from selected users				
10		5	20						
<input type="checkbox"/>	ID ↑ ↓	N° reviews ↑ ↓	Average score ↑ ↓	Variance score ↑ ↓					
<input type="checkbox"/>	A00177463W0XWB16A900 5	13	4.385	0.544					
<input type="checkbox"/>	A022899328A0QOR3ZDC T	10	3.300	2.610					
<input type="checkbox"/>	A04309042SDSL8YX2HRR 7	5	3.800	0.960					
<input type="checkbox"/>	A068255029AHTHDZURN U	9	4.333	0.889					
<input type="checkbox"/>	A06944682TFWOKV4GJK X	9	4.778	0.173					
<input type="checkbox"/>	A1004703RC78J9	6	3.667	0.889					
<input type="checkbox"/>	A1006HCQDMYC5W	9	4.333	1.111					
<input type="checkbox"/>	A1008DPSPKC6J	7	4.286	0.776					
<input type="checkbox"/>	A100DXV4SLAMP	7	5.000	0.000					
<input type="checkbox"/>	A100MUAHGQCF6	5	4.200	2.560					
<input type="checkbox"/>	A109L91863SLUQ	9	4.667	0.889					
<input type="checkbox"/>	A100VQNP6I54HS	15	5.000	0.000					

Figura 3: Griglia utente

Una volta creato il modello, non sarà più possibile crearne un altro a meno che non si utilizzi il tasto: *ResetModel*.

Dashboard

Reset model

N° words	10
N° topics	5
N° iterations	20
Selected users	A00177463W0XWB16A9O05

Figura 4: Reset Model

Vista Prodotto

Le viste più interessanti sono quelle che vengono mostrate una volta che il modello è stato creato. Se si visualizza un prodotto, si possono trovare le parole che lo caratterizzano ed un interessante grafico riguardante la distribuzione dei topic su quel prodotto:

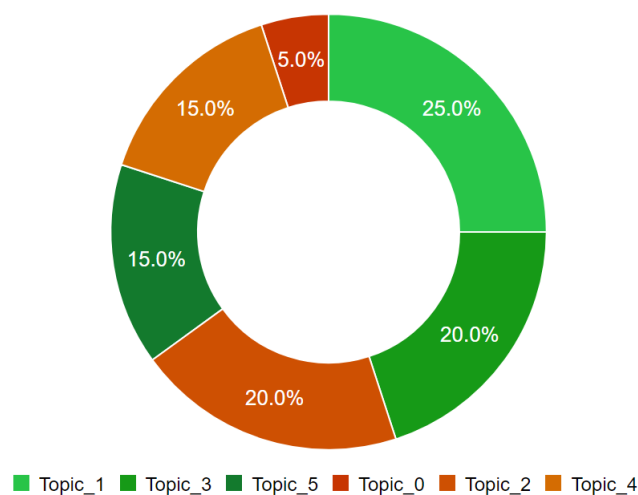


Figura 5: Reset Model

Vista Utente

Se si ispeziona il singolo utente, si possono visualizzare le parole che più lo caratterizzano (sia positivamente che negativamente) oltre che un grafico a torta che illustra come sono

distribuiti gli *score* delle recensioni date.

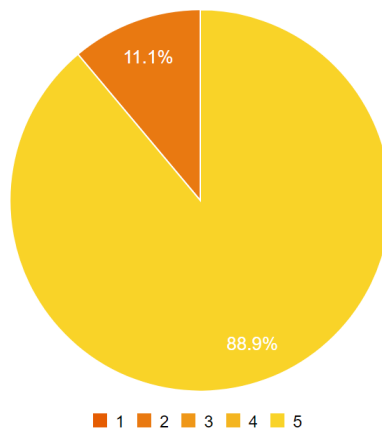


Figura 6: Reset Model

Il grafico principale dell'intera applicazione è il grafico che illustra l'andamento dei topic prediletti dall'utente in base allo scorrere del tempo. Sono stati creati 2 grafici che mostrano l'andamento dei topic positivi

Positive topics

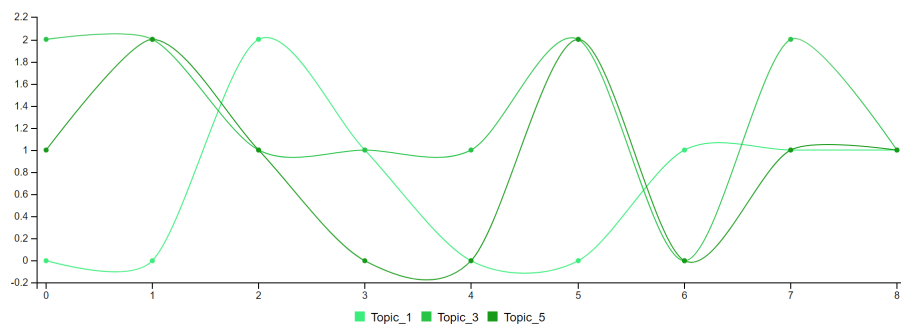


Figura 7: Reset Model

e di quelli negativi

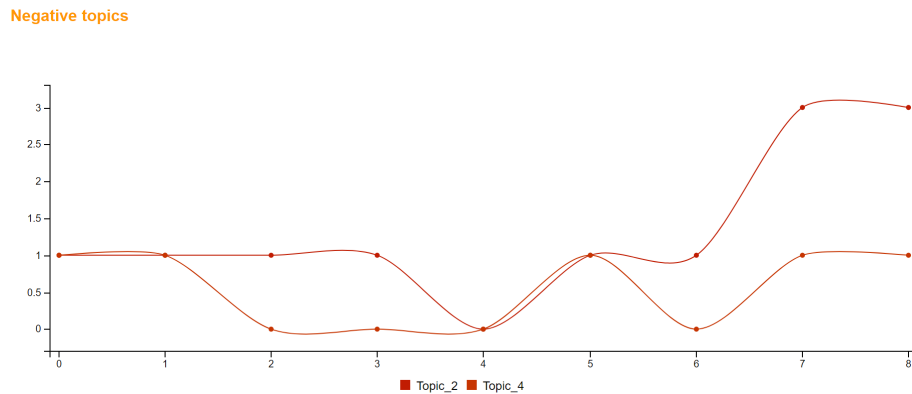


Figura 8: Reset Model



Dai grafici in fig.6 e fig.7 si evince che l'utente abbia perso interesse nel *topic 1* (positivo) durante le sue recensioni, lasciando il posto al *topic 2* (negativo).

Conclusione

Dopo aver creato diversi modelli, si è giunti alla conclusione che vi sono diversi fattori che possono portare alla creazione di un modello accettabile:

- la corretta valorizzazione degli iperparametri α , β e γ : sono stati provati numerosi valori per questi iperparametri. La combinazione migliore, dovuta anche a varie ricerche fatte, vede la valorizzazione dei rispettivi parametri a: 2.5, 0.3 e 0.1.
- il numero di topic selezionati: questo valore è veramente difficile da impostare. Si può pensare che più ci siano recensioni, più vadano impostati topic nel modello: è vero in tutti i casi? Se vengono impostati troppi topic, si rischia di avere un modello molto dispersivo (parole ripetute, topic simili), dove è difficile capire la distribuzione effettiva dei commenti dell'utente.
- poche parole per topic: questo rischio è strettamente legato al sistema ideato per individuare la distribuzione di topic nel tempo. Più il topic conterrà poche parole, più sarà difficile trovare un'intersezione fra esse e quelle dell'utente - prodotto.
- numero di iterazioni: essendo un modello probabilistico, più iterazioni vengono fatte su di esso maggiore è la precisione nel rilevare *sentiment* e *topic*.

Un problema non indifferente è stata la scelta del linguaggio usato per lo sviluppo di *JST*. L'ottimizzazione della memoria di *Python* e le sue performance sono di gran lunga differenti rispetto ad un linguaggio compilato (es. *C++*). Questo ha portato a dover

ridurre il numero di iterazioni per poter restare in tempi accettabili di elaborazione, oltre alle problematiche dovute alla memorizzazione di matrici troppo grosse (`sparse_matrix` in `array`). Esistendo un algoritmo *JST* ben sviluppato in *C++*, si potrebbe in futuro usare un *wrapping* per poter importare quell' algoritmo in *Python*