Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in
Statistica per le Tecnologie e le Scienze

RELAZIONE FINALE

Text Mining su recensioni italiane di TripAdvisor

Relatore Prof. Livio Finos
Dipartimento di Scienze Statistiche

Laureando: Giovanni Corradini
Matricola N. 1124705

Anno Accademico 2017/2018

Indice

Introduzione

È stato stimato[1][2][3] che da gennaio 2017 a gennaio 2018 il numero di utilizzatori internet in tutto il mondo è passato da 3.77 miliardi a 4.25 miliardi (su 7.59 miliardi di persone, pari alla popolazione umana), ovvero è aumentato del 7% in un solo anno. In italia, solo nel 2006, il 63% delle persone non aveva mai utilizzato internet mentre oggi ben il 71.7% delle famiglie italiane accede regolarmente alla rete[4].
Non c'è dubbio quindi che internet sia sempre più popolare e diffuso e che con la vastissima quantità e varietà di informazioni e servizi che eroga stia generando una mole di contenuti che aumenta con una velocità elevatissima. Si stima[5] infatti che internet produca un flusso di informazioni pari a 150 exabyte al mese, di cui l'80% è in forma non-strutturata, ossia senza una struttura predefinita com'è, per esempio, il "classico" dataframe con osservazioni per riga e varibaili per colonna; di questo 80%, il 90% è composto da materiale video e il restante 10% ("solo" 15 milioni di terabyte) da materiale in forma testuale. Le principali "miniere" di dati tesuali facilmente accessibili da chiunque in rete sono i social network, i blog, i forum, i siti delle principali internet companies, i portali web a altro ancora.
Quindi, data la spaventosa crescita di produzione di informazioni testuali nel mondo ed anche in Italia, è stato scelto di produrre questo lavoro di tesi sull'analisi statistica di testi (qui esclusivamente in lingua italiana), anche nota come text mining. I dati scelti per l'analisi sono le recensioni italiane scritte sul portale web di viaggi TripAdvisor che, come dice Wikipedia, *'Con più di 35 milioni di recensioni e una media di 29 milioni di visitatori al sito ogni mese sin dal suo lancio, TripAdvisor è il più grande sito di viaggi sul web'*; in particolare i testi (assieme ad altre caratteristiche relative ad ogni recensione) sono stati estratti dal sito mediante tecniche di web scraping con *python* ed è stato scelto di prendere le recensioni di due fra gli hotel italiani con più recensioni in italiano: l'hotel A.Roma Lifestyle Hotel di Roma e l'hotel Palazzo Caracciolo Napoli MGallery by Sofitel di Napoli.
Dopo aver estratto le recensioni degli hotel, essendo il dato testuale una tipologia di dato non strutturato, per poterle analizzare è stato necessario in primo luogo preprocessarle, ovvero pulirle, normalizzarle e correggerle, e in seguito dare loro una struttura; in questo caso quella del *tidy text format*:[6] *'a table with one token per row'*, ovvero un dataframe che presenta un *token* (spesso una parola) distinto per ogni riga. Una volta che i testi hanno assunto una struttura *consistente*[7], è possibile applicare su di essi una serie di metodologie di analisi testuale esplorativa, come la decostruzione del testo in una bag-of-words utile per varie analisi grafiche e per l'analisi della frequenza di parole e n-grammi.
Oltre all'analisi esplorativa testuale è stata compiuta, utilizzando ancora i testi strutturati in modo 'ordinato', una sentiment analysis basata su un dizionario (ontologico) di sentimenti italiani. I risultati dell'analisi esplorativa testuale e della sentiment analysis sono stati usati successivamente come punto di partenza per la fase di modellazione; in particolare è stata

[1] *InternetWorldStats*https://www.internetworldstats.com/stats.htm)
[2] *Statista*https://www.statista.com/topics/1145/internet-usage-worldwide/
[3] *WeAreSocial e Hootsuite*
[4] *Istat*
[5] *Guida al text mining e alla sentiment analysis con R*
[6] *Robinson D.,Silge J.,Text Mining with R,Sebastopol,O'Reilly,2017,1*
[7] *Robinson D.,Silge J.,Text Mining with R,Sebastopol,O'Reilly,2017,1*

utilizzata una foresta casuale per classificare le persone che hanno scritto una recensione riguardante uno dei due hotel in base alle *stelle* (valutazione da 1 a 5 allegata alla recensione scritta), sfruttando le variabili estrapolate dal sito di TripAdvisor e altre create a partire da esse.
Questo studio è suddiviso in tre sezioni : la prima parte è dedicata alla fase di raccolta e sistemazione dei dati, nella seconda ci si focalizza sulle tecniche di text mining e sentiment analysis applicate ai testi delle recensioni e nella terza viene trattata la fase di classificazione delle osservazioni.
R è il linguaggio di programmazione scelto per compiere le varie analisi dei dati, in particolare sono stati usati i pacchetti contenuti nel 'core' del TidyVerse (*ggplot2, tidyr, dplyr, tibble, readr, purrr*) e altri specifici per il text mining (come *tidytext* e *widyr*) che lavorano in armonia con i *"tidy principles"*. Oltre a questi pacchetti è stato utilizzato, in sincrono con *tidytext*, *TextWiller*, pacchetto per il text mining appositamente dedicato alla lingua italiana e *randomForestSRC* per la fase di classificazione.

Capitolo 1

Raccolta e sistemazione dati

A differrenza di molte fra le più grandi industrie e software house del mondo come Microsoft, Facebook e Google, TripAdvisor non eroga pubblicamente le sue API (Application Programming Interface), specialmente se il loro utilizzo è per fini statistici[8]: quindi per il reperimento dei dati relativi alle recensioni italiane degli hotel scelti è stato necessario utilizzare tecniche di *web scraping*, in quanto non è stato possibile ricevere un database delle recensioni 'già pronto' (in formato HTML o JSON) semplicemente inviando una richiesta HTTP all'API di TripAdvisor.
La prima fase del lavoro di tesi è stata quindi quella di ricercare manualmente su TripAdvisor gli hotel italiani con più recensioni fra tutti gli hotel italiani sul sito, con la maggioranza delle recensioni scritte in lingua italiana e con testi di lunghezza sufficiente a produrre analisi statistiche testuali consistenti. Sono stati dunque individuati due hotel pertinenti alle caratteristiche di ricerca: l'hotel *A.Roma Lifestyle* di Roma e l'hotel *Palazzo Caracciolo Napoli* di Napoli i quali, sebbene siano solamente due hotel, assieme totalizzano poco più di 2500 recensioni, che è sembrato un buon compromesso per la numerosità campionaria in quanto si hanno allo stesso tempo abbastanza testi per poter riconoscere gli argomenti specifici per ognuno dei due hotel, ma non troppi testi per far sì che l'onere computazionale delle varie analisi fosse abbastanza contenuto.
Successivamente sono stati estratti, mediante tecniche di web scraping in linguaggio *python*, i testi delle recensioni ed altre variabili ad essi collegate, in particolare la valutazione da 1 a 5 dell'hotel e la data in cui questi sono stati scritti. Infine, prima di procedere con le analisi statistiche testuali, è stato necessario preprocessare i dati raccolti e dare loro una struttura conforme a quella richiesta dai pacchetti *R* del *tidyverse*.

[8] *Request API Access, TripAdvisor*

1.1 Web Scraping

Il 'web scraping', in passato anche noto come *screen scraping*, *data mining*, *web harvesting* o *web data extracting* è una tecnica informatica di estrazione di dati compiuta scrivendo un programma che invia automaticamente richieste ad un sito web e che colleziona solo una limitata e specifica quantità di informazioni (il programma effettua un parsing sui dati collezionati)[9][10]. Il web scraping si differenzia dal 'web crawling' per il fatto che in quest'ultimo non avviene una selezione delle informazioni, ma bensì il 'crawler' (il bot che effettua il web crawling) visita tutti i link di un sito web, acquisisce e indicizza le parole presenti in tutti i documenti del sito visitato e le raggruppa in un database; tuttavia le due tecniche non sono scollegate, infatti *'to do web scraping, you have to do some degree of web crawling to move around the websites'*[11].
Dato che, come si vedrà nei seguenti capitoli, il preprocessing dei dati e tutte le analisi compiute su di essi (in particolare le analisi svolte sui testi relativi alle recensioni) sono state realizzate con il programma *R*, anche il reperimento delle recensioni degli hotel scelti è stato tentato inizialmente con *rvest*, pacchetto *R* creato appunto per il web scraping (o *web harvesting*). Però, dato l'eccessivo divario tra la semplicità di utilizzo di *rvest* e la complessità della struttura della pagina web di TripAdvisor (per esempio il pacchetto non riesce a scaricare il contenuto integrale dei testi delle recensioni troppo lunghe), non è stato possibile effettuare lo scraping delle pagine di TripAdvisor con il pacchetto *R*, ed è stato necessario quindi utilizzare un altro programma; perciò, soprattutto grazie alla disponibilità di pacchetti sofisticati e specifici per il web scraping, e per motivi che saranno chiariti a breve, la scelta del linguaggio di programmazione per l'estrazione dei dati è ricaduta su *python*.

[9] *Ryan M.,Web Scraping with Python,Sebastopol,O'Reilly,2015,7*
[10] *Wikipedia: web scraping*
[11] *Quora: web scraping and web crawling*

1.1.1 Python e BeautifulSoup

Python è un linguaggio di programmazione open-source ad alto livello, *supporta diversi paradigmi di programmazione, come quello object-oriented (con supporto all'ereditarietà multipla), quello imperativo e quello funzionale*[12], ed offre moltissime librerie per lavori di quasi qualsiasi varietà e complessità: in particolare *Beautiful Soup* è una libreria specifica per estrarre dati in formato HTML e XML (libreria per fare web scraping per l'appunto).
Fortunatamente, sul sito *towardsdatascience*, è disponibile il codice scritto in *python* (utilizzando *Beautiful Soup*) da Susann Li, Sr. Data Scientist di Toronto dello stesso sito, per fare web scraping sulla pagina web di TripAdvisor; perciò, dopo aver reperito il codice su Github[13], è bastato adattarlo alla pagina italiana del sito e alle recensioni degli hotel individuati in precedenza (codice in appendice) per estrapolare, oltre che ai testi delle recensioni, la data (giorno-mese-anno) in cui sono state scritte, la valutazione 'in stelle' (da 1 a 5) dell'hotel ed anche, se presenti, il titolo che può essere aggiunto alla recensione, l'user_name di chi scrive la recensione, i contributi e voti utili che una persona con l'account su TripAdvisor ha collezionato nella sua esperienza recensistica, il luogo dove è stata scritta la recensione e un'ultima variabile dicotomica introdotta per distinguere i due hotel, in quanto il codice per fare 'scraping' è stato progettato per estrapolare le informazioni di un solo hotel alla volta.

[12] *Cos'è Python?*
[13] *Github: NLP-with-Python*

1.2 Procedure preliminari

Dopo aver estrapolato ed organizzato le recensioni, è stato necessario conferire ai dati una struttura conforme a quella richiesta dal *tidyverse*, per poi dunque preprocessarli (o *standardizzarli*), ovvero normalizzare e pulire i testi, e rifinire la forma di alcune variabili.

1.2.1 Standardizzazione dei dati

In primo luogo i due database degli hotel di Roma e di Napoli, precedentemente salvati in formato csv, sono stati letti con la funzione *read_csv*: questo sia perchè di default la funzione legge i testi con la codifica di caratteri Unicode UTF-8, che non crea alcun tipo di problema di traduzione (al contrario della ISO-8859-1 che è il default di *R* e di *R Markdown*), sia perchè crea un oggetto di tipo *tibble*.

Lettura dei database estrapolati da TripAdvisor

```
dfr <- read_csv('C:/Users/zebra/Desktop/Roma.csv')
dfn <- read_csv('C:/Users/zebra/Desktop/Napoli.csv')
dim(dfr)#1090 righe
dim(dfn)#1621 righe
```

La *tibble* è un formato più moderno del *data.frame* di *R*, che, oltre ad avere una rapidità di calcolo maggiore, ha il vantaggio che su di essa possono essere applicate facilmente le funzioni di tutti i pacchetti del *tidyverse* e del pacchetto *tidytext*, utilizzate in questo capitolo per la fase di perprocessing dei dati e nel prossimo per l'analisi dei testi. Inoltre la *tibble*, specialmente in presenza di database contenenti testi, dispone di una visualizzazione più compatta e ordinata rispetto al *data.frame*, il che permette una comprensione più immediata dei dati (Tabella 1).

Table 1: Alcune variabili del database pre-preprocessing

	review_title	review_date	rating
1	Top	23 settembre 2018	50
2	Un mondo a parte	23 settembre 2018	40
3	Tutto al top	22 settembre 2018	50
4	Ideale per convention	21 settembre 2018	50
5	Ottimo hotel Ristorante eccezionale	20 settembre 2018	50
6	Business	16 settembre 2018	50
7	O ti rilassi o lavori!	16 settembre 2018	40
8	soggiorno perfetto...lusso comfort e relax	15 settembre 2018	50
9	serata ai sapori dal mondo	14 settembre 2018	40
10	Beautiful style!	13 settembre 2018	50

Sono state dunque create le due *tibble* contenenti i dati, facendole successivamente confluire, unendole in una unica. Vista la maggiore complessità del dato testuale rispetto ad un dato di tipo numerico, per alleggerire gli oneri computazionali delle varie analisi e per produrre analisi più efficaci sui testi, è stato eseguito il preprocessing dei testi. Inoltre sono state create variabili a partire da quelle estrapolate da TripAdvisor (come il gender e le variabili derivate dalla data) che saranno utilizzate nella sezione dedicata alla modellazione.
Il processo di standardizzazione dei dati si è svolto nelle seguenti fasi:

- sui testi delle recensioni (la variabile review_body) è stata applicata la funzione *normalizzaTesti* del pacchetto *TextWiller*: questa funzione racchiude in sé le funzioni *normalizzaslang*, *normalizzapunteggiatura*, *normalizzahtml*, *normalizzaemote* e *normalizzacaratteri* che assieme operano una pulizia del testo (correggendo errori di battitura ed eliminando punteggiatura, spazi vuoti e siti web), una normalizzazione vera e propria dei testi che consiste nel trasformare il linguaggio diffuso su internet e i modi di dire in un italiano più 'standard' (italiano 'da vocabolario', affinchè possa essere trattato in maniera più agevole dall'algoritmo per lo 'stemming' che verrà utilizzato in seguito), una traduzione delle emoticons in stringhe che le rappresentano (per esempio 'XD' viene tradotta in 'EMOTEAMAZE') e la conversione di tutti i caratteri in minuscolo

- sono state applicate funzioni dei pacchetti *tm* e *stringr* per ulteriori pulizie dei testi, come la rimozione dei numeri e di alcuni caratteri speciali, la correzione di alcuni errori di battitura frequenti e l'accorpamento di parole con lo stesso significato ma scritte in maniera diversa (per esempio 'qualità' e 'qualitá' sono state accorpate in 'qualità'). Inoltre, per semplicità di utilizzo, i livelli della variabile dicotomica indicante a quale dei due hotel appartiene la recensione sono stati ricodificati in 'Roma' e 'Napoli'

- alcune funzioni del pacchetto *lubridate* sono state applicate alla data (la variabile review_date) per la conversione del suo tipo (da $< carattere >$ a $< data >$) e per creare variabili a partire da essa, come il giorno della settimana e il giorno del mese

- è stata creata la variabile dicotomica gender, che rappresenta il sesso della persona che scrive la recensione, mediante l'impiego della funzione *classificaUtenti* di *TextWiller* che, applicata ai nomi utenti (user_name) delle persone con un profilo su TripAdvisor (poco meno del 10% del campione), riesce a classificarli in 'masc' e 'femm'

- la funzione *classificaUtenti* è stata riutilizzata sulla variabile user_location, che rappresenta il luogo dove è stata scritta la recensione (presente in circa il 15% delle recensioni), ma con il parametro aggiuntivo 'vocabolarioLuoghi' (database presente in *Textwiller* focalizzato sulle città italiane) così da classificare tutte le città in cui sono state scritte le recensioni in 6 aree geografiche: Centro, Estero, Isole, Nord-est, Nord-ovest e Sud; anche questa variabile verrà utilizzata come predittore nella fase di classificazione

- sono stati apportati piccoli accorgimeti alla *tibble* modificata, come l'aggiunta di una colonna indicante l'ID dell'osservazione, utile nel seguente capitolo quando i testi delle recensioni verranno decomposti in parole.

Tutte le funzioni utilizzate per il preprocessing sono state applicate in maniera 'consistente' sulla *tibble* (Tabella 2), ovvero mantenendo una struttura con una riga per ogni osservazione e una colonna per ogni variabile, grazie alle funzioni *mutate* e *select* del pacchetto *dplyr*.

Table 2: Alcune variabili del database post-preprocessing

	user_location	data	stelle	titolo
1	Sud	2018-09-23	5	Top
2	Centro	2018-09-23	4	Un mondo a parte
3	Sud	2018-09-22	5	Tutto al top
4		2018-09-21	5	Ideale per convention
5		2018-09-20	5	Ottimo hotel Ristorante eccezionale
6		2018-09-16	5	Business
7	Sud	2018-09-16	4	O ti rilassi o lavori!
8	Centro	2018-09-15	5	soggiorno perfetto...lusso comfort e relax
9		2018-09-14	4	serata ai sapori dal mondo
10	Sud	2018-09-13	5	Beautiful style!

1.2.2 Stop words

L'ultima fase di quello che ormai non è più preprocessing dei dati ma solamente un'azione preliminare ad alcune analisi, è stata la preparazione (e non ancora la rimozione dai testi) della lista di stop words: le stop words sono parole vuote, poco significative, che compaiono molto spesso all'interno delle frasi e che solitamente si presentano in forma di preposizione, avverbio o congiunzione. Per fare ciò è stata modificata *itastopwords*, la lista contenente le più frequenti stop words italiane presente in *TextWiller*: in essa sono state introdotte alcune parole frequenti nelle recensioni che non erano d'interesse per l'analisi ed è stata trasformata in una *tibble* per poterla utilizzare agevolmente assieme al database preprocessato.
Visto che nei capitoli seguenti saranno compiute delle analisi che richiedono tecniche anche molto diverse tra loro, non è stato applicato l'usuale stemming (riduzione delle parole alla loro radice) e non sono state eliminate le stop words dai testi in quanto, prima di poter applicare queste procedure di preprocessing, è necessario 'tokenizzare' (decomporre ogni testo in una *bag of words*) i testi delle recensioni; infatti, le tecniche per alcune analisi richiedono il testo non decomposto, altre suddiviso in parole e altre ancora suddiviso in n-grammi (gruppi di n parole adiacenti).

Capitolo 2

Text Mining e Sentiment Analysis: un approccio *tidy*

2.1 Analisi Esplorativa Testuale

Avendo ora a disposizione i dati (e anche la lista di stop words) con una struttura *tidy*, è possibile manipolarli, modellarli e visualizzarli facilmente[14]. In particolare si possono applicare i metodi principali del pacchetto *tidytext* (e di tutti gli altri pacchetti *tidy* come *dplyr* e *ggplot2*) sui testi delle recensioni e sui titoli allegati alla maggior parte di esse; questi metodi comprendono la decomposizione del testo in una bag of words, il conteggio delle frequenze delle parole nei testi e l'analisi dei bigrammi. I vantaggi principali derivati dal disporre di una *tibble* in formato *tidy* stanno nella semplicità, varietà e velocità delle tecniche di analisi effettuabili su di essa. Infatti, come si vedrà nel corso di questo capitolo, la *tibble* contenente le variabili verrà decomposta e ricomposta più volte, ed ogni volta verrà manipolata in maniera diversa, preservando sempre la sua struttura iniziale.

2.1.1 Analisi delle frequenze

Prima di svolgere l'analisi vera e propria, sono state analizzate numericamente alcune variabili al di fuori dei testi, per avere una idea di base della costituzione del dataset. Alcune statistiche descrittive relative ad alcune variabili presenti nel dataset sono riportate in Tabella 3.

Table 3: Summary iniziale di alcune variabili dei due hotel

	hotel	num_rec	prima_rec	ultima_rec	media_stelle
1	Roma	1090	2016-02-08	2018-09-23	4.633
2	Napoli	1621	2009-12-08	2018-09-22	4.356

[14] *Wickham H.,Tidy Data,Journal of Statistical Software,Vol 86,2018*

Il processo di analisi esplorativa testuale si è svolto nelle seguenti fasi:

- è stata applicata una 'tokenizzazione' alle recensioni ed ai titoli, ovvero *i testi sono stati suddivisi in unità minime di analisi dette "token"*[15]. Inizialmente l'unità minima di analisi (un token) è stata posta pari ad una parola
- successivamente alla decomposizione dei testi di ogni recensione e di ogni titolo in una *bag-of-words* mediante la funzione *unnest_tokens* del pacchetto *tidytext*, è stato possibile rimuovere le stop words da essi mediante la funzione *anti_join* del pacchetto *dplyr*
- prima di conteggiare le parole presenti nei testi, è stato fatto un confronto tra i testi prima e dopo la fase di preprocessing (inclusa l'eliminazione delle stop words); è emerso che prima del preprocessing il numero di parole distinte tra loro presenti nell'insieme delle recensioni era circa 19700. Successivamente il numero di parole è sceso a 13734: ciò ha portato ad una notevole riduzione della complessità del problema, nonchè ad una diminuzione del 'rumore'. I testi dei titoli sono stati invece solamente 'tokenizzati', e non preprocessati, sia in quanto già molto brevi e riassuntivi, sia perché spesso sono composti solamente da due/tre parole che, combinate assieme, sono un modo di dire od una tipica affermazione
- dopo aver filtrato le stop words, sono state conteggiate le frequenze delle parole più usate nei testi e nelle recensioni (Tabella 4).

Table 4: Parole più frequenti tra i titoli e le recensioni

	parola_titolo	n_titolo	parola_recensione	n_recensione
1	ottimo	244	non	2976
2	napoli	224	molto	2766
3	soggiorno	121	personale	1595
4	molto	102	colazione	1339
5	eccellente	98	napoli	1212
6	non	95	camera	1196
7	ottima	94	bello	996
8	bello	92	camere	925
9	relax	81	ristorante	813
10	spa	81	centro	802
11	bellissimo	76	struttura	776

[15] *Unipa: tokenizzazione*

Una rappresentazione alternativa alla tabella o all'istogramma per la visualizzazione delle parole presenti in un testo è la wordcloud (nuvola di parole). Questa ha la peculiare caratteristica di attribuire un font di dimensioni più grandi alle parole più frequenti e questo fa della wordcloud una lista pesata[16].

Wordcloud: quando un disegno parla più di mille parole… o quasi

```
df_tok %>%
  count(parola) %>%
  with(wordcloud(parola, n, max.words = 200))
```

Figura 1: Wordcloud

[16] *Wikipedia: Nuvola di etichette*

Dopo aver conteggiato le parole più frequenti tra i testi dei titoli e delle recensioni, sono stati analizzati anche i bigrammi più frequenti in essi, ovvero tutte le coppie di parole adiacenti presenti nei testi (n-grammi con n=2).
È stato scelto di analizzare le coppie di parole adiacenti per capire il contesto in cui si trovano molte parole 'ambigue', ovvero molte parole che se poste prima o dopo di determinati termini assumono un significato diverso o cambiano il significato di questi termini; i risultati delle varie analisi (in particolare grafiche) saranno utilizzati come punto di partenza per la sentiment analysis, oggetto di studio della prossima sezione.
Per suddividere il testo in bigrammi ci si è serviti nuovamente della funzione *unnest_tokens*, con la modifica relativa ai parametri *token = "ngrams"* e *n = 2*. Successivamente, è stato necessario 'spezzare' i bigrammi per poter rimuovere le stop words. Anche con i bigrammi è stato scelto di rimuoverle solo dalle recensioni e non dai titoli. Infine, sono stati conteggiati i bigrammi più frequenti (Tabella 5).

Table 5: Bigrammi più frequenti tra i titoli e le recensioni

	parola_titolo	n_titolo	parola_recensione	n_recensione
1	molto bello	30	molto bello	351
2	week end	27	centro storico	283
3	centro storico	26	personale molto	237
4	palazzo storico	19	palazzo caracciolo	191
5	qualità prezzo	19	personale gentile	139
6	rapporto qualità	19	qualità prezzo	136
7	ottima posizione	18	rapporto qualità	126
8	bella struttura	16	molto gentile	125
9	molto buono	16	minuti piedi	115
10	ottima struttura	15	stazione ferroviaria	106
11	ottimo soggiorno	13	palazzo storico	102
12	ottima scelta	12	molto disponibile	98
13	5 stelle	11	cortile interno	96
14	ottimo rapporto	10	consiglio vivamente	95
15	ottima spa	9	gentile disponibile	95

2.1.2 *non-bigrammi* e stemming

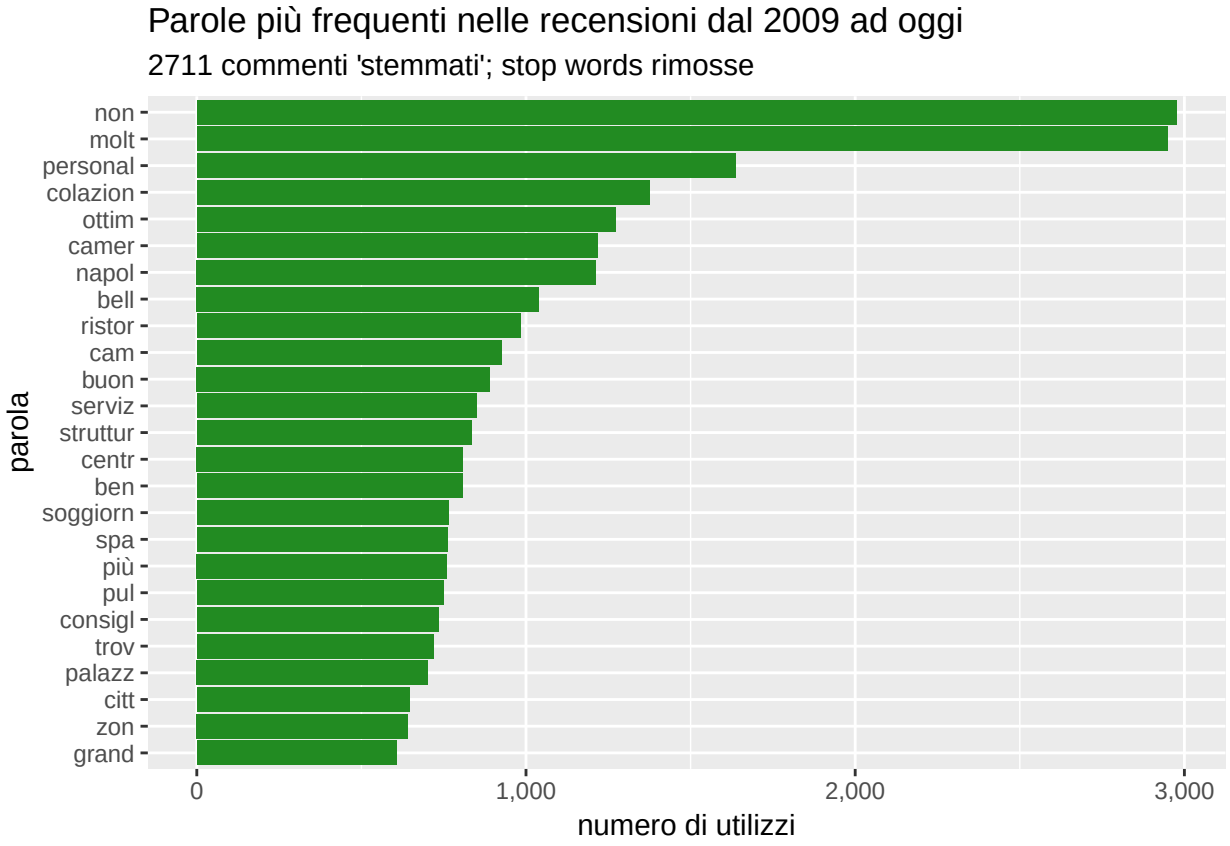
Dopo aver conteggiato i bigrammi più frequenti, sia tra le recensioni che tra i titoli, è stato deciso di analizzare nello specifico le coppie di parole che hanno come prima parola *'non'*, parola non stop word più frequente tra le recensioni 'pre-stemming', con quasi 3000 occorrenze. La scelta è dovuta al fatto che se certi sentimenti, ossia parole specifiche con valenza sentimentale (alle quali sarà assegnato un valore numerico nel paragrafo seguente), fossero preceduti dalla parola (*'non'*), assumerebbero un significato completamente diverso, portando così ad una loro errata classificazione, attuata mediante le tecniche di sentiment analysis basate sul dizionario di sentimenti.
Nei titoli non sono molto frequenti i 'non-bigrammi' (Tabella 6); tuttavia emerge il fatto che spesso le parole che seguono *'non'* sono parole indicative di un'opinione riguardante l'hotel. Per questo motivo alcuni non-bigrammi frequenti sono stati aggiunti successivamente al vocabolario dei termini negativi, utilizzato poi per la sentiment analysis.

```
pre_non_title <- bigrammi_filtrati_title %>%
  filter(parola1 == "non") %>%
  count(parola1, parola2, sort = TRUE)
```

Table 6: Parole più frequenti nei titoli precedute da "non"

	parola1	parola2	n
1	non	all'altezza	6
2	non	molto	4
3	non	perdere	4
4	non	centrale	3
5	non	caro	2
6	non	dimenticare	2
7	non	lasciatevi	2
8	non	male	2
9	non	preoccupatevi	2
10	non	tanto	2

È stato poi effettuato un conteggio delle parole più frequenti tra le recensioni (istogramma qui sotto), ma dopo aver applicato loro uno stemming, ovvero il processo di riduzione della forma flessa di una parola alla sua forma radice, detta tema (o appunto 'stem' in inglese)[17]. L'applicazione dello stemming è stata realizzata, sempre in sincrono con le funzioni *tidy mutate* e *select*, mediante la funzione *wordStem* del pacchetto *SnowballC* che provvede uno stemmer molto efficiente per la lingua italiana; infatti, dopo aver applicato lo stemming ed aver rimosso le stop words dai testi, le parole distinte tra loro presenti nelle recensioni si sono quasi dimezzate, passando da 13734 a 7581.



[17] *Wikipedia: Stemming*

2.2 Sentiment Analysis

Quando si parla di sentiment analysis (anche nota come *opinion mining*) ci si riferisce a quell'insieme di tecniche che identificano, estraggono, quantificano e studiano l'opinione di un soggetto rispetto ad un certo argomento, per capire se è favorevole o contrario a qualche tema o per dare una polarità positiva/negativa, od un punteggio, al suo 'stato emotivo/sentimentale'[18][19]. In questo studio le tecniche di sentiment analysis saranno applicate ai testi delle recensioni e dei titoli, ma il loro utilizzo non si limitata solamente a questo, infatti queste tecniche possono essere utilizzate anche, per esempio, trattando immagini o dati biometrici.
Oltre alla grande quantità di argomenti che tratta, la sentiment analysis offre un'altrettanta varietà di tecniche con la quale può essere compiuta. In questo studio ci si concentrerà sulla tecnica basata sul vocabolario di sentimenti, ovvero sarà utilizzata una lista di parole etichettate come positive o negative (i sentimenti) che verranno confrontate con quelle presenti nelle recensioni per vedere, nel complesso, quante parole positive o negative sono contenute nelle recensioni; lo scopo di ciò è quello di classificare le recensioni in positive o negative in base alla prevalenza di sentimenti positivi o negativi in esse.
L'analisi è stata effettuata con tecniche simili alle precedenti, e con gli stessi pacchetti *R*; inoltre il dizionario di sentimenti utilizzato è un adattamento, al contesto di TripAdvisor, di quello presente in *TextWiller*: *vocabolariMadda*.
L'obiettivo di questa sentiment analysis, oltre a quello di riuscire ad identificare (corettamente) il maggior numero di sentimenti nei testi, è quello di creare delle variabili, come il numero di sentimenti positivi e negativi presenti nelle recensioni, da utilizzare come predittori nella fase di classificazione delle recensioni in base alle stelle, a cui sarà dedicato il prossimo capitolo.

2.2.1 Modifica del dizionario di sentimenti

L'analisi esplorativa testuale dei paragrafi precedenti ha evidenziato le parole ed i bigrammi più importanti delle recensioni e dei titoli (oltre che alle parole più frequenti precedute da 'non'): questi risultati sono serviti come punto di partenza per la modifica del vocabolario dei sentimenti. Infatti, dopo aver sistemato alcuni sentimenti del dizionario che in questo contesto risultavano erroneamente classificati (ad esempio la parola 'super', molto utilizzata su TripAdvisor, non ha una valenza negativa), sono stati introdotti nel dizionario i sentimenti (parole post-stemming) ed i bigrammi (senza aver applicato loro lo stemming) più frequenti nelle recensioni, che non erano presenti nel vocabolario. Il dizionario, dopo essere stato adattato (la manipolazione delle stringhe è stata effettuata ancora una volta con le funzioni del pacchetto *stringr*), come da prassi è stato convertito in una *tibble* affinchè potesse agire in conformità coi principi del *tidyverse*. In aggiunta è stata creata una copia modificata del dizionario, con dei punteggi pari a -1 e 1 al posto dei sentimenti negativi e positivi rispettivamente, che risulterà utile in alcuni contesti.

[18] *Wikipedia: Sentiment Analysis*
[19] *Guida al text mining e alla sentiment analysis con R*

2.2.2 Sentiment Analysis con dizionario ontologico

Per compiere l'analisi con il dizionario, essendo quest'ultimo composto non solo da parole ma anche da bigrammi, è stato necessario inizialmente 'tokenizzare' il testo in coppie di parole, poi identificare i bigrammi presenti sia nelle recensioni che nel dizionario ('bi-sentimenti' in Tabella 7) ed infine salvarli a parte, associandoli a 'positive' o 'negative' in base al sentimento che esprimono.

Conteggio bigrammi collegati ai sentimenti nelle recensioni

```
cont_bi_sent <- bigrammi_filtrati %>%
  unite(parola, parola1, parola2, sep = "-") %>%
  inner_join(voc_sent_score) %>%
  count(parola, score, sort = TRUE) %>%
  ungroup()
```

Table 7: Bi-Sentimenti più frequenti tra le recensioni

	parola	score	n
1	molto-bello	1	351
2	personale-gentile	1	139
3	molto-gentile	1	125
4	molto-disponibile	1	98
5	consiglio-vivamente	1	95
6	non-molto	-1	91
7	ottima-colazione	1	73
8	colazione-ottima	1	71
9	molto-buono	1	69
10	unica-pecca	-1	69

Per poter individuare correttamente le parole 'post-stemming' relative ai sentimenti rimanenti nelle recensioni, conteggiate in Tabella 8, è stato necessario eliminare i bigrammi collegati ai sentimenti individuati precedentemente.
Anche per le tecniche di sentiment analysis basate sul vocabolario dei sentimenti l'applicazione su *R* è stata resa più agevole dall'utilizzo dei pacchetti *tidy dplyr* e *tidyr*, che hanno permesso di scomporre, ricomporre e manipolare più volte la *tibble* delle recensioni ed il dizionario senza dover ricorrere a codici eccessivamente articolati.

Conteggio parole collegate ai sentimenti nelle recensioni

```
cont_rev_st <- dati %>%
  unnest_tokens(parola, recensione) %>%
  mutate(parola = wordStem(parola, 'italian')) %>%
  anti_join(ita_stop_words, by = 'parola') %>%
  inner_join(voc_sent_score) %>%
  count(parola, score, sort = TRUE) %>%
  ungroup()
```

Table 8: Sentimenti più frequenti tra le recensioni

	parola	score	n
1	ottim	1	1272
2	bell	1	1039
3	buon	1	891
4	ben	1	808
5	pul	1	750
6	grand	1	607
7	dispon	1	601
8	bellissim	1	539
9	gentil	1	504
10	cur	1	450

Una volta compiuta la sentiment analysis sui testi delle recensioni, il procedimento è stato ripetuto sui titoli, conteggiati in Tabella 8, sfruttando però esclusivamente le parole 'post-stemming', in quanto il numero di bigrammi collegati ai sentimenti presenti nei titoli era decisamente piccolo.
Infine, le parole ed i bigrammi collegati ai sentimenti individuati nelle recensioni e nei titoli, accompagnati dall'ID dell'osservazione a cui si riferiscono, indispensabile per creare i predittori che verranno utilizzati nel prossimo capitolo, sono stati uniti in un'unica *tibble* per avere i conteggi complessivi di tutti i sentimenti presenti.

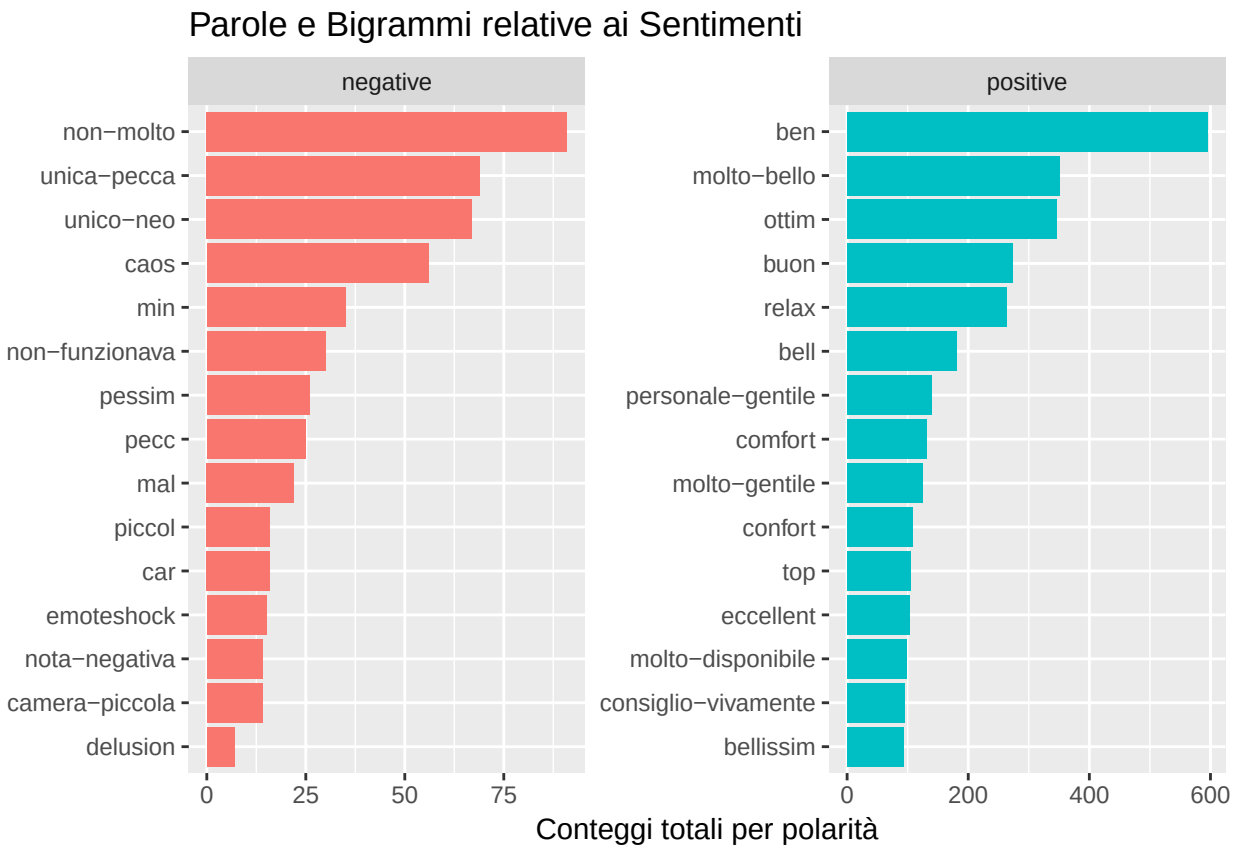
Table 9: Sentimenti più frequenti tra i titoli

	parola	score	n
1	ottim	1	345
2	bell	1	138
3	buon	1	106
4	eccellent	1	104
5	bellissim	1	94

Visualizzazione delle parole 'post-stemming' collegate ai sentimenti più frequenti tra le recensioni con una ComparisonCloud: la grandezza delle parole è proporzionale alla loro frequenza.

```
dati %>%
  unnest_tokens(parola, recensione) %>%
  mutate(parola = wordStem(parola, 'italian')) %>%
  anti_join(ita_stop_words, by = 'parola') %>%
  inner_join(vocab_sent) %>%
  count(parola, sentimento, sort = TRUE) %>%
  acast(parola ~ sentimento, value.var = 'n', fill = 0) %>%
  comparison.cloud(colors = c('red2', 'blue2'), max.words = 200)
```

Dopo aver individuato e conteggiato le parole ed i bigrammi presenti nelle recensioni e nei titoli, e dopo aver salvato ciascun gruppo in una *tibble* differente, essi sono stati fatti confluire in un'unica *tibble*, associando ciascun elemento di ogni gruppo al relativo ID, così da poter conteggiare complessivamente quanti sentimenti sono presenti nei testi, quali sono quelli più frequenti e la loro polarità (vedi istogrammi sottostanti).



L'analisi ha prodotto risultati interessanti: si nota infatti la ben definita distinzione tra sentimenti e polarità. Inoltre, definendo lo score di ogni recensione come somma tra sentimenti positivi e negativi presenti in essa (valore 1 e -1 rispettivamente), è emerso che la media delle stelle per le 50 recensioni con lo score più alto e più basso è, rispettivamente, 4.56 e 3.42, stante ad indicare una plausibile relazione tra sentimenti e stelle.
Utilizzando metodi basati sul vocabolario di sentimenti, non si riesce però a misurare l'adeguatezza di queste tecniche, se non per via grafica o approssimativa; è stato dunque deciso di utilizzare i risultati della sentiment analysis per costruire delle variabli che verranno utilizzate come predittori durante la fase di modellazione statistica.

Capitolo 3

Classificazione delle recensioni

Nella parte finale di questo studio è stato scelto di classificare le recensioni in base alle stelle, ossia alla valutazione dell'hotel da 1 a 5 che ogni persona fornisce quando scrive la recensione. Visto che nel campione vi è solamente l'1% del totale delle persone che ha dato 1 stella di valutazione, il 2% che ha dato 2 stelle ed il 6% che ha dato 3 stelle, è stato deciso di accorpare queste tre classi in una unica, etichettandola come 'male'. Le altre due classi, relative alla categoria di persone che hanno dato 4 e 5 stelle di valutazione, sono state invece etichettate rispettivamente come 'buono' (31%) e 'ottimo' (60%).
Come modello di classificazione è stato scelto la *foresta casuale*, in particolare l'algoritmo proposto da Breiman e Cutler nel 2001, che è stato applicato alle recensioni di TripAdvisor mediante la funzione *rfsrc* del pacchetto *randomForestSRC*, che permette inoltre la gestione di valori mancanti, presenti in qualche variabile del dataset.

3.1 Foresta casuale

Un albero binario di decisione è un modello utilizzato in statistica e in 'data mining' per la classificazione e per la regressione delle osservazioni. Questo metodo si basa sulla segmentazione dello spazio dei predittori in regioni sempre più semplici, alle quali corrispondono le unità statistiche con una combinazione dei predittori appartenente a quella regione. Più precisamente, l'albero rappresenta l'insieme delle regole utilizzate per suddividere lo spazio dei predittori in J regioni distinte che non si sovrappongono. Ogni nodo indica 'dove', ovvero rispetto a quale predittore e in quale valore tra quelli che esso assume, è stato applicato lo 'split' (la suddivisione), mentre le foglie dell'albero indicano la classe di appartenenza delle unità statistiche. L'algoritmo per la crescita dell'albero, partendo dal primo nodo, per ogni nodo (J), applica lo split sul valore del predittore che minimizza il coefficiente di Gini ($G = \sum_{k=0}^{K-1} \widehat{p_{jk}}(1 - \widehat{p_{jk}})$, con $\widehat{p_{jk}}$ pari alla proporzione di unità appartenenti alla k-esima classe nella regione J) per quel nodo e non si ferma fino a che non raggiunge un valore soglia, per esempio fino a che l'albero non raggiunge una certa profondità. In questo caso il coefficiente di Gini, proposto per la prima volta dallo statistico italiano nel 1912, assume il ruolo di indice di impurità di un nodo. Dopo aver fatto crescere l'albero, ogni foglia viene etichettata con il nome della classe che presenta maggiore frequenza fra tutte le classi della foglia. In altre parole, tutte le osservazioni con una determinata combinazione di predittori, che le accomuna nella medesima foglia, verranno classificate in base alla classe che va di moda in quella foglia. Dopo aver fatto crescere l'albero, solitamente più del dovuto, lo si deve potare, ovvero si rimuovono i rami e le foglie che non soddisfano un certo criterio di costo-complessità; in questo caso si tagliano i rami e le foglie senza i quali non si avrebbe un eccessivo aumento del coefficiente di Gini, penalizzato per il numero di foglie nell'albero, questo perchè, mantenendo un albero più folto del dovuto, si possono riscontrare problemi di eccessivo adattamento del modello ai dati. L'albero di decisione è uno strumento che offre molti vantaggi, tra cui la semplicità di comprensione e rappresentabilità, il fatto che può essere utilizzato con tutti i tipi di variabili (anche in presenza di valori mancanti) e la presenza di algoritmi estremamente rapidi per la classificazione.

Tra i difetti di questo metodo troviamo che può soffrire di elevata variabilità se fatto crescere eccessivamente, mentre se fatto crescere troppo poco può essere affetto da un'ingente distorsione e perciò l'accuratezza delle sue previsioni, anche se fatte su un insieme di osservazioni sensibilmente diverso da quello di allenamento, spesso non è sufficientemente adeguata.
Un valido metodo per ridurre questa variabilità e di conseguenza migliorare la capacità predittiva dell'albero di decisione è il 'bagging', diminutivo di *bootstrap aggregating*, proposto per la prima volta da Breiman nel 1994. Questo procedimento consiste nel creare un numero B di replicazioni bootstrap del campione iniziale (si creano B campioni, con B sull'ordine delle migliaia, della stessa numerosità del campione di partenza costruiti mediante campionamento con reinserimento dal campione di partenza) e nell'adattare un singolo albero di decisione per ogni campione bootstrap. Dopo aver ottenuto B alberi di decisione, si allocano le unità statistiche nelle classi che ricevono più 'voti' fra tutti gli alberi (ogni unità viene allocata nella classe che va di moda per quell'unità fra tutti i B alberi). Essendo lo scopo del bagging la riduzione della variabilità dell'albero, questo metodo viene applicato 'aggregando' alberi non potati, ossia alberi ad elevata varianza e distorsione ridotta.
La foresta casuale, attuata tramite l'algoritmo proposto da Breiman e Cutler nel 2001, è uno sviluppo del bagging che si basa sul suo stesso principio di costruzione: a differenza di quest'ultimo la foresta casuale, durante la crescita di ogni albero, per scegliere lo split ottimale da effettuare su ogni nodo di ognuno dei B alberi non chiama in causa tutti i p predittori, ma solo una parte $m < p$ di essi, scelta in maniera casuale. Questa variazione del bagging rimuove l'eventuale correlazione presente tra le coppie di predittori, implicando un'ulteriore riduzione della variabilità dell'albero di decisione con un conseguente aumento dell'accuratezza delle sue previsioni[20]. Inoltre, dato che ogni albero è adattato su un campione bootstrap differente, per avere una stima non distorta dell'accuratezza della previsione di tali metodi non è necessario suddividere il campione iniziale in un insieme di allenamento ed in un insieme di verifica, oppure utilizzare procedure di validazione incrociata. Infatti, nello stimare ognuno dei B alberi sugli altrettanti campioni bootstrap, viene utilizzata solo una parte del campione di partenza, escludendo l'altra che rimane tagliata fuori; questa parte, chiamata *'out-of-bag'* del campione, viene appunto utilizzata come surrogato dell'insieme di verifica per la stima dell'errore di previsione del modello.
Rispetto al singolo albero di decisione ci sono dei miglioramenti dal punto di vista della stabilità e dell'accuratezza dei risultati, ma anche dei peggioramenti relativi alla minor interpretabilità di essi; disponendo infatti di un insieme di alberi (una foresta appunto) e non di uno soltanto, non è possibile avere un'immediata rappresentazione visiva del modello. Di conseguenza, per misurare "l'importanza" che ogni variabile ha avuto nella costruzione del modello è stato utilizzato un indice opportuno, definito dalla media calcolata sui B campioni bootstrap della riduzione totale che ogni variabile ha apportato all'indice di impurità di Gini (durante la crescita di ogni albero, per ogni nodo).

[20] *Breiman L.,RANDOM FORESTS,Berkeley,2001,5*

3.2 Applicazione a TripAdvisor

Prima della fase di classificazione svolta mediante la foresta casuale è stato modificato il dataset utilizzato nelle analisi dei capitoli precedenti: sono state introdotte molte nuove variabili create a partire da quelle estrapolate da TripAdvisor e da altre create in seguito alla sentiment analysis, come la variabile 'score' che rappresenta la somma dei sentimenti presenti tra i titoli e le recensioni (ottenuta utilizzando il dizionario dei sentimenti con *'negative = -1'* e *'positive = 1'*). Altre variabili sono state create a partire da interazioni tra la variabile 'score' e le variabili 'contributi', 'user_location' e 'voti utili'; altre sono state invece definite come 'score'/'numero di parole' e 'score'/'numero di caratteri' per ogni recensione; altre ancora sono state create a partire dalle interazioni tra le variabili estrapolate dalla data, come il giorno della settimana o del mese, e le variabili 'score' e 'numero di parole per ogni recensione'.
Alla fine della fase di preprocessing antecedente alla modellazione, dopo aver accorpato i livelli della variabile 'stelle' in 'male', 'buono' e 'ottimo', per poter attuare la classificazione con le funzioni del pacchetto *randomForestSRC*, è stato necessario trasformare la *tibble*, composta da 2711 righe e 45 variabili, in un *data.frame* delle stesse dimensioni. La scelta del pacchetto è ricaduta su *randomForestSRC* principalmente perchè, impostando nella funzione *rfsrc* il parametro *'na.action = "na.impute"'*, nella fase di costruzione della foresta casuale i valori mancanti vengono 'imputati', ossia sostituiti con la media dei valori non mancanti presenti in quella variabile (se essa è quantitativa) o con la moda dei valori (se essa è qualitativa).
Gli altri iperparametri impostati nel modello sono:

- il numero di alberi *ntree*, che rappresenta il numero B di replicazioni bootstrap del campione, impostato pari a 1000 in quanto anche con un numero elevato di alberi non si corre il rischio di eccessivo adattamento[21][22]
- il numero m di variabili *mtry*, tra le p presenti, che è stato posto pari a 7 (circa $\sqrt{p}$[23]), valore che minimizza l'errore di previsione del modello
- *'nodesize = 1'*, che sta ad indicare che il numero di osservazioni presenti in ogni foglia di ogni albero deve essere almeno pari a uno (come detto in precedenza, utilizzando metodi di aggregazione di alberi si tende a preferire alberi molto grandi)
- *'nsplit = 0'*, il quale sta a significare che, nella scelta del valore ottimale per lo split, viene calcolato l'indice di impurità di Gini per tutti i valori del predittore scelto e non solo per una sua porzione selezionata casualmente
- *'importance = TRUE'*, che serve a riportare, nel summary finale del modello, l'importanza che ogni variabile ha avuto nella costruzione della foresta
- *'ensemble = "oob"'*, ovvero che non è richiesta la costruzione di un insieme di verifica per stimare in maniera non distorta l'errore di previsione 'o-o-b' del modello, definito come la media, fatta su tutte le osservazioni del campione, delle proporzioni di alberi

[21] *R documentation : randomForestSRC::rfsrc*
[22] *James G.,Witten D.,Hastie T.,Tibshirani R.,An Introduction to Statistical Learning,New York,2013,320*
[23] Breiman L.,Cutler A.,*Random Forests*

che classificano erroneamente un'osservazione quando essa è nella parte 'out-of-bag' del campione[24].

Codice Foresta Casuale

```
set.seed(16)
rf_clf <- rfsrc(stelle~., data = db,
              ntree = 1000,
              mtry =  7,
              nodesize = 1,
              nsplit = 0,
              importance = TRUE,
              ensemble = "oob",
              bootstrap = "by.root",
              samptype = "swr",
              na.action = "na.impute",
              nimpute = 1)
```

I coefficienti relativi all'importanza di ogni variabile, presenti nel summary della funzione *rfsrc* (non riportato), hanno mostrato che le variabili 'gender', 'data' e quelle ad esse collegate nella costruzione del modello, non hanno apportato significative riduzioni del coefficiente di Gini rispetto agli altri predittori. Risulta inoltre che la variabile 'score' e la variabile 'score'/'numero di parole per ogni recensione' hanno un'importanza maggiore rispetto a tutti gli altri predittori. È stato dunque scelto di riadattare la foresta casuale alle osservazioni, eliminando i 28 predittori meno importanti. Dopo questa operazione l'errore di previsione non si è ridotto notevolmente; questo è in parte dovuto al fatto che la foresta casuale applica una sorta di selezione automatica delle variabili e perciò i predittori scartati nel secondo modello erano tenuti già poco in considerazione nel primo.
L'accuratezza del modello, ovvero il rapporto tra unità predette correttamente e unità totali predette, stimata sul campione 'o-o-b', è risultata pari al 66%. Per capire se i risultati della sentiment analysis, integrati nel modello sotto forma di variabili, hanno apportato a quest'ultimo dei validi miglioramenti, è stata adattata una foresta casuale sulle stesse osservazioni su cui è stato adattato il modello precedente, utilizzando però solamente i predittori originariamente estrapolati da TripAdvisor. Il modello senza i predittori derivanti dalla sentiment analysis classifica correttamente circa 250 unità statistiche in meno rispetto al modello comprensivo di questi predittori, fornendo una stima dell'accuratezza pari a 57%.

[24]Breiman L.,Cutler A.,*Random Forests*

Conclusione

Come visto in questo lavoro di tesi, la varietà di tecniche relative al *text mining* presenti nei pacchetti *R* utilizzati è stata fondamentale per poter maneggiare agevolmente le recensioni estrapolate da TripAdvisor, in particolare i testi contenuti in esse. Infatti, grazie alla grande duttilità delle tecniche utilizzate, è stato molto comodo integrare i *'bi-sentimenti'* nella sentiment analysis con il dizionario ontologico, azione che ha procurato notevoli progressi a questa tecnica.
I predittori introdotti nel modello mediante le tecniche di text mining e sentiment analysis, hanno apportato buoni miglioramenti alla procedura di classificazione: in particolare, la stima dell'accuratezza delle previsioni della foresta casuale è aumentata di nove punti percentuali rispetto al modello orfano di questi predittori. Nonostante questo aumento di prestazioni del modello non ci si può tuttavia ritenere pienamente soddisfatti dei risultati finali: la stima dell'accuratezza del modello finale si assesta infatti a 66%, stando a significare che vi è qualche problema durante la fase di classificazione. Indagando le possibili cause di ciò si è notata la marcata differenza tra il numero di recensioni negative ed il numero di recensioni positive (in favore di quest'ultime); inoltre, le variabili disponibili subito dopo la fase di web scraping ('data', 'user_location', 'contributi', etc) non sono particolarmente utili per descrivere la variabilità del campione.
Un possibile sviluppo da apportare al modello, con lo scopo di migliorarne le prestazioni, riguarda l'applicazione, durante la fase di costruzione dei campioni bootstrap sui quali verranno adattati gli alberi della foresta casuale, di una procedura di downsampling alle classi più numerose del campione, in maniera tale da avere alberi più bilanciati. Un'altra possibile strada da intraprendere predilige invece l'utilizzo del 'boosting' (Schapire, 1990) come algoritmo di classificazione, in quanto esso pondera i dati ad ogni nuova iterazione, considerando maggiormente le classi poco numerose.

Appendice

Codice Python per il web scraping

```
import requests
from bs4 import BeautifulSoup
import csv
import webbrowser
import io
def display(content, filename='output.html'):
    with open(filename, 'wb') as f:
        f.write(content)
        webbrowser.open(filename)
def get_soup(session, url, show = False):
    response = session.get(url)

    if show:
        display(response.content, 'temp.html')
    if response.status_code != 200 :
        print('codice di risposta:', response.status_code)

    else:
        return BeautifulSoup(response.text, 'html.parser')
def scrape(url):
    session = requests.Session() #creazione sessione per mantenere
                                 #tutti i cookie ed altri dati
                                 #legati alla sessione tra richieste
    session.headers.update({
            'User-Agent':
            'Mozilla/5.0 (X11; Ubuntu;Linux x86_64;rv:57.0) Firefox/57.0',
        })
    items = parse(session, url)
    return items
def post_soup(session, url, params, show=False):
    '''Read HTML from server and convert to Soup'''
    r = session.post(url, data=params)

    if show:
        display(r.content, 'temp.html')
    if r.status_code != 200: # not OK
        print('[post_soup] status code:', r.status_code)
    else:
        return BeautifulSoup(r.text, 'html.parser')
def parse(session, url):
    print('PARSING: ', url)
```

```
    soup = get_soup(session, url)

    if not soup:
        print('no Soup: ', url)
        return
    num_Rec=soup.find('span', class_='reviews_header_count').text
    num_Rec=num_Rec[1:-1]
    num_Rec=num_Rec.replace('.', '')
    num_Rec=int(num_Rec)
    print('N recensioni: ', num_Rec)
    #PER SCORRERE LE RECENSIONI
    #Si suppone che l'URL originale non contenga -orX-
    url_Template = url.replace('.html', '-or{}.html')
    items=[]
    offset = 0
    while(True):
        sub_url = url_Template.format(offset)
        sub_items=parse_reviews(session, sub_url)
        if not sub_items:
            break
        items += sub_items

        if len(sub_items) < 5:
            break
        offset += 5
    return items
def get_reviews_ids(soup):
    items = soup.find_all('div', attrs={'data-reviewid': True})
    if items:
        reviews_ids = [x.attrs['data-reviewid'] for x in items][::2]
        print('[get_reviews_ids] data-reviewid:', reviews_ids)
        return reviews_ids
def get_more(session, reviews_ids):
    url = 'https://www.tripadvisor.it/Jax?Mode=EXPANDED_HOTEL_REVIEWS'
    payload = {
        'reviews': ','.join(reviews_ids), # ie. "577882734,577547902",
        #'contextChoice': 'DETAIL_HR'
        'widgetChoice': 'EXPANDED_HOTEL_REVIEW_HSX',
        'haveJses': 'earlyRequireDefine,templates-dust-en_US,taevents',
        'haveCsses': 'apg-Hotel_Review-in',
        'Action': 'install',
    }
    soup = post_soup(session, url, payload)
    return soup
```

```
'''
funzione per salvare le recensioni ed i dati correlati
'''
def parse_reviews(session, url):

    soup = get_soup(session, url)

    if not soup:
        print('no SOUP: ', url)
        return
    hotel_name = soup.find('h1', id='HEADING').text
    reviews_ids = get_reviews_ids(soup)
    if not reviews_ids:
        return
    soup = get_more(session, reviews_ids)
    if not soup:
        print('[parse_reviews] no soup:', url)
        return
    items = []
    for idx, review in enumerate(soup.find_all('div', class_='Selector')):
        badgets = review.find_all('span', class_='badgetext')
        if len(badgets) > 0:
            contributions = badgets[0].text
        else:
            contributions = '0'
        if len(badgets) > 1:
            helpful_vote = badgets[1].text
        else:
            helpful_vote = '0'
        user_loc = review.select_one('div.userLoc strong')
        if user_loc:
            user_loc = user_loc.text
        else:
            user_loc = ''

        bubble_rating = review.select_one('span.ui_bubble_rating')['class']
        bubble_rating = bubble_rating[1].split('_')[-1]
        item = {
            'hotel_name': hotel_name,
            'review_title': review.find('span', class_='noQuotes').text,
            'review_body': review.find('p', class_='partial_entry').text,
            'review_date': review.find('span',class_='ratingDate')['title'],
            'contributions': contributions,
            'helpful_vote' : helpful_vote,
```

```
            'user_name': review.find('div', class_='info_text').text[5:-6],
            'user_location': user_loc,
            'rating': bubble_rating
        }
        items.append(item)
        print('\n--- review ---\n')
        for key,val in item.items():
            print(' ', key, ':', val)
    print()
    return items
def write_in_csv(items, filename='results.csv',
                 headers=['hotel_name', 'review_title', 'review_body',
                          'review_date', 'contributions', 'helpful_vote',
                          'user_name' , 'user_location', 'rating'],
                 mode='w'):
    print('--- CSV ---')
    with io.open(filename, mode, encoding="utf-8") as csvfile:
        csv_file = csv.DictWriter(csvfile, headers)
        if mode == 'w':
            csv_file.writeheader()
        csv_file.writerows(items)

start_urls = [
    'https://www.tripadvisor.it/A_Roma_Lifestyle_Hotel-Rome_Lazio.html',
    'https://www.tripadvisor.it/Palazzo_Caracciolo-Naples_Campania.html'
]
for url in start_urls:
    # get all reviews for 'url' and 'lang'
    items = scrape(url)
    if not items:
        print('No reviews')
    else:
        # write in CSV
        filename = url.split('Reviews-')[1][:-5]
        print('filename:', filename)
        write_in_csv(items, filename + '.csv', mode='w')
```

Preprocessing dei dati

```
data("vocabolarioLuoghi")

dati <- dati %>%
  rowid_to_column('ID') %>%
  mutate(recensione = normalizzaTesti(as.character(review_body))) %>%
  select(-review_body) %>%
  transform(review_date = dmy(review_date)) %>%
  mutate(data = review_date) %>%
  select(-review_date) %>%
  mutate(stelle = rating/10) %>%
  select(-rating) %>%
  mutate(hotel = as_factor(hotel_name)) %>%
  select(-hotel_name) %>%
  mutate(titolo = review_title) %>%
  select(-review_title) %>%
  mutate(voti_utili = helpful_vote) %>%
  select(-helpful_vote) %>%
  mutate(contributi = contributions) %>%
  select(-contributions)

dati <- dati %>%
   mutate(user_name = classificaUtenti(user_name)) %>%
  mutate(gender = ifelse(user_name %in% c('masc', 'femm'),
                      as.factor(user_name), NA)) %>%
  select(-user_name) %>%
  mutate(user_location =
           classificaUtenti(user_location, vocabolarioLuoghi)) %>%
  mutate(user_location =
           ifelse(user_location %in%
                    levels(as.factor(vocabolarioLuoghi$categoria)),
                    user_location, NA))

dati$hotel <- fct_recode(dati$hotel,
                         Roma = 'A.Roma Lifestyle Hotel',
                         Napoli =
                           'Palazzo Caracciolo Napoli MGallery by Sofitel')

dati$recensione <- removeNumbers(dati$recensione)
```

Modifica e salvataggio di itastopwords

```
data("itastopwords")
itastopwords <- c(itastopwords,'check','altro','altra','altri','altre',
                  'quasi','out', 'poichè', 'siccome', 'mentre',
                  'affinchè', 'prima', 'dopo', 'nonostante', 'malgrado',
                  'benchè', 'qualora', 'purchè', 'inoltre', 'dunque',
                  'anzi', 'tuttavia', 'oppure', 'infatti', 'sebbene',
                  'ecco', 'qui', 'qua', 'ancora', 'solo', 'sempre',
                  'proprio', 'certo', 'appunto', 'pure', 'ormai',
                  'abbastanza', 'adesso', 'praticamente', 'ovviamente',
                  'certamente','chiaramente', 'giustamente',
                  'immediatamente', 'direttamente', 'particolarmente',
                  'specialmente', 'dopo', 'dentro', 'sotto',
                  'prima', 'circa', 'davanti', 'appena', 'invece',
                  'oltre', 'indietro', 'intanto', 'spesso', 'presto',
                  'hotel', 'albergo', 'stato', 'stat', 'stata', 'stati',
                  'state', 'uno', 'due', 'tre', 'cinque')

ita_stop_words <- as.tibble(itastopwords) %>%
  mutate(parola = value) %>%
  select(-value) %>%
  arrange(parola)

write_csv(ita_stop_words, 'C:/Users/zebra/Desktop/ita_stop_words.csv')
```

Tokenizzazione, rimozione stop words e conteggio parole nelle recensioni

```
df_tok <- dati %>%
  select(recensione) %>%
  unnest_tokens(parola, recensione) %>%
  anti_join(ita_stop_words, by = "parola")
conteggio_parole <- df_tok %>%
  count(parola, sort = TRUE)
```

Tokenizzazione e conteggio delle parole nei titoli

```
df_tok_title <- dati %>%
  select(titolo) %>%
  unnest_tokens(parola, titolo) %>%
  anti_join(ita_stop_words, by = "parola")
conteggio_parole_title <- df_tok_title %>%
  count(parola, sort = TRUE)
```

Analisi dei bigrammi presenti nelle recensioni

```
bigrammi <- dati %>%
  unnest_tokens(bigramma, recensione, token = "ngrams", n = 2)
bigrammi_separati <- bigrammi %>%
  separate(bigramma, c("parola1", "parola2"), sep = " ")
bigrammi_filtrati <- bigrammi_separati %>%
  filter(!parola1 %in% ita_stop_words$parola) %>%
  filter(!parola2 %in% ita_stop_words$parola)
conteggio_bigrammi <- bigrammi_filtrati %>%
  unite(bigramma, parola1, parola2, sep = ' ') %>%
  count(bigramma, sort = TRUE)

bigrammi_title <- dati %>%
  unnest_tokens(bigramma, titolo, token = "ngrams", n = 2)
bigrammi_separati_title <- bigrammi_title %>%
  separate(bigramma, c("parola1", "parola2"), sep = " ")

bigrammi_filtrati_title <- bigrammi_separati_title %>%
  filter(!parola1 %in% ita_stop_words$parola) %>%
  filter(!parola2 %in% ita_stop_words$parola) %>%
  filter(!is.na(parola1)) %>%
  filter(!is.na(parola2))

conteggio_bigrammi_title <- bigrammi_filtrati_title %>%
  unite(bigramma, parola1, parola2, sep = ' ') %>%
  count(bigramma, sort = TRUE)
```

Unione delle due liste di sentimenti in un'unica tibble

```
data("vocabolariMadda")
vocab_sent <- tibble(
  parola = c(vocabolariMadda$negative, vocabolariMadda$positive),
  sentimento = c(rep('negative', length(vocabolariMadda$negative)),
                 rep('positive', length(vocabolariMadda$positive)))
)
```

Eliminazione sentimenti positivi e negativi classificati erroneamente

```
err_pos <- c('pur', 'qui', 'not', 'san', 'fin', 'super', 'port', 'ver', 'sal' )
vocab_sent <- filter(vocab_sent, !parola %in% err_pos)
err_neg <- c('bar', 'tram', 'super', 'cup', 'for', 'set', 'pass', 'port', 'lasc',
             'lung', 'incred', 'mond','lasc', 'lung', 'per', 'ritorn', 'sud')
vocab_sent <- filter(vocab_sent, !parola %in% err_neg)
```

Introduzione nuovi sentimenti positivi nel dizionario

```
vocab_sent <- vocab_sent %>%
  add_row(parola = 'personale-gentile', sentimento = 'positive') %>%
  add_row(parola = 'adeguato', sentimento = 'positive') %>%
  add_row(parola = 'molto-vantaggioso', sentimento = 'positive') %>%
  add_row(parola = 'valido', sentimento = 'positive') %>%
  add_row(parola = 'non-malee', sentimento = 'positive') %>%
  add_row(parola = 'ottimale', sentimento = 'positive') %>%
  add_row(parola = 'ottimalee', sentimento = 'positive') %>%
  add_row(parola = 'davvero-eccellente', sentimento = 'positive') %>%
  add_row(parola = 'sempre-ottimalee', sentimento = 'positive') %>%
  add_row(parola = 'camera-pulita', sentimento = 'positive') %>%
  add_row(parola = 'ottima-posizione', sentimento = 'positive') %>%
  add_row(parola = 'posizione-centrale', sentimento = 'positive') %>%
  add_row(parola = 'molto-gentile', sentimento = 'positive') %>%
  add_row(parola = 'molto-comodo', sentimento = 'positive') %>%
  add_row(parola = 'wifi-gratuito', sentimento = 'positive') %>%
  add_row(parola = 'colazione-ottima', sentimento = 'positive') %>%
  add_row(parola = 'molto-pulita', sentimento = 'positive') %>%
  add_row(parola = 'molto-pulite', sentimento = 'positive') %>%
  add_row(parola = 'camera-grande', sentimento = 'positive') %>%
  add_row(parola = 'consiglio-vivamente', sentimento = 'positive') %>%
  add_row(parola = 'molto-bello', sentimento = 'positive') %>%
  add_row(parola = 'molto-disponibile', sentimento = 'positive') %>%
  add_row(parola = 'ottima-colazione', sentimento = 'positive') %>%
  add_row(parola = 'bellissim', sentimento = 'positive') %>%
  add_row(parola = 'molto-buono', sentimento = 'positive') %>%
  add_row(parola = 'relax', sentimento = 'positive')
```

Introduzione nuovi sentimenti negativi nel dizionario

```
vocab_sent <- vocab_sent %>%
  add_row(parola = 'non-grandissimo' , sentimento = 'negative') %>%
  add_row(parola = 'non-funzionava' , sentimento = 'negative') %>%
  add_row(parola = 'non-frigobar' , sentimento = 'negative') %>%
  add_row(parola = 'non-bidet' , sentimento = 'negative') %>%
  add_row(parola = 'malee' , sentimento = 'negative') %>%
  add_row(parola = 'pessim' , sentimento = 'negative') %>%
  add_row(parola = 'camera-piccola' , sentimento = 'negative') %>%
  add_row(parola = 'camera-piccolina' , sentimento = 'negative') %>%
  add_row(parola = 'unica-pecca' , sentimento = 'negative') %>%
  add_row(parola = 'mancanza-bidet' , sentimento = 'negative') %>%
  add_row(parola = 'senza-bidet' , sentimento = 'negative') %>%
  add_row(parola = 'nota-negativa' , sentimento = 'negative') %>%
  add_row(parola = 'non-ben' , sentimento = 'negative') %>%
  add_row(parola = 'non-molto' , sentimento = 'negative') %>%
  add_row(parola = 'unico-neo' , sentimento = 'negative') %>%
  add_row(parola = 'non-consiglio' , sentimento = 'negative') %>%
  add_row(parola = "non-all'altezza" , sentimento = 'negative') %>%
  add_row(parola = 'piccol' , sentimento = 'negative')
```

Salvataggio del dizionario dei sentimenti

```
write_csv(vocab_sent, 'C:/Users/zebra/Desktop/vocab_sent.csv')
```

Costruzione del dizionario con i punteggi

```
voc_sent_score <- tibble(
  parola = (arrange(vocab_sent, sentimento))$parola,
  score = c(rep(-1, sum(vocab_sent$sentimento == 'negative')),
            rep(1, sum(vocab_sent$sentimento == 'positive')))
  )
```

Salvataggio del dizionario con i punteggi

```
write_csv(voc_sent_score, 'C:/Users/zebra/Desktop/vocab_sent_score.csv')
```

Bibliografia

1. Branca M.,*Strategie di Sentiment Analysis: confronti e nuove proposte*, Relazione finale,Università di Padova,Facoltà di Scienze Statistiche, 2014.
2. Breiman L.,*RANDOM FORESTS*,Berkeley,2001.
3. Grolemund G.,Wickham H.,*R for Data Science*,Sebastopol,O'Reilly,2017.
4. James G.,Witten D.,Hastie T.,Tibshirani R.,*An Introduction to Statistical Learning*,New York,2013.
5. Porcu V.,*Guida al text mining e alla sentiment analysis con R*,Italia,Hoepli,2016.
6. Robinson D.,Silge J.,*Text Mining with R*,Sebastopol,O'Reilly,2017.
7. Ryan M.,*Web Scraping with Python*,Sebastopol,O'Reilly,2015.
8. Wickham H.,*Tidy Data*,Journal of Statistical Software,Vol 86,2018

Sitografia

1. Breiman L.,Cutler A.,*Random Forests*: Cutlerhttps://www.stat.berkeley.edu/~breiman/RandomForests/
2. *Cos'è Python?*: https://www.python.it/about/
3. *Github: NLP-with-Python*: https://github.com/susanli2016/NLP-with-Python/
4. *InternetWorldStats*: https://www.internetworldstats.com/stats.htm
5. *Istat*: https://www.istat.it/it/archivio/216672
6. *Quora: web scraping and web crawling*: https://www.quora.com/
7. *R Documentation, rfsrc*: https://www.rdocumentation.org/packages/randomForestSRC/versions/2.7.0/topics/rfsrc
8. *Request API Access, TripAdvisor*: https://developer-tripadvisor.com/content-api/request-api-access/
9. *Ryan M.,Web Scraping with Python,Sebastopol,O'Reilly,2015,7*: https://yanfei.site/docs/dpsa/references/PyWebScrapingBook.pdf
10. *Statista*: https://www.statista.com/topics/1145/internet-usage-worldwide/
11. *Unipa: tokenizzazione*: http://www1.unipa.it/sorce/didattica/sei1213/SEI1213_05_tokenizzazione.pdf
12. *WeAreSocial e Hootsuite*: https://wearesocial.com/it/blog/2018/01/global-digital-report-2018
13. *Wikipedia*: https://it.wikipedia.org/wiki/

Pacchetti R utilizzati

1. Kuhn M.,Wing J.,Weston S.,Williams A.,Keefer C.,Engelhardt A.,Cooper T.,Mayer Z.,Kenkel B.,il core team di R,Benesty M.,Lescarbeau R.,Ziem S.,Scrucca L.,Tang Y.,Candan C.,Hunt T.,*caret: Classification and Regression Training*,2018,{https://CRAN.R-project.org/package=caret}
2. Pedersen T.,*ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*,2018,{https://CRAN.R-project.org/package=ggraph}
3. Csardi G.,Nepusz T.,*The igraph software package for complex network research*,2006,{http://igraph.org}
4. Grolemund G.,Wickham H.,*Dates and Times Made Easy with lubridate*,2011,{http://www.jstatsoft.org/v40/i03/}
5. Ishwaran H.,Kogalur U.B.,*Random Forests for Survival, Regression, and Classification (RF-SRC)*,2018,{https://cran.r-project.org/package=randomForestSRC}
6. Wickham H.,*scales: Scale Functions for Visualization*,2018,{https://CRAN.R-project.org/package=scales}
7. Bouchet-Valat M.,*SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*,2014,{https://CRAN.R-project.org/package=SnowballC}
8. Hlavac M.,*stargazer: Well-Formatted Regression and Summary Statistics Tables*,2018,{https://CRAN.R-project.org/package=stargazer}
9. Solari D.,Sciandra A.,Rinaldo M.,Redaelli M.,Finos L.,*TextWiller: Collection of functions for text mining, specially devoted to the italian language*,2016,{: https://github.com/livioivil/TextWiller}
10. Silge J.,Robinson D.,*tidytext: Text Mining and Analysis Using Tidy Data Principles in R*,2016,{http://dx.doi.org/10.21105/joss.00037}
11. Wickham H.,*tidyverse: Easily Install and Load the 'Tidyverse'*,2017,{https://CRAN.R-project.org/package=tidyverse}
12. Feinerer I.,Hornik k.,*tm: Text Mining Package*,2008,{http://www.jstatsoft.org/v25/i05/}
13. Fellows I.,*wordcloud: Word Clouds*,2018,{https://CRAN.R-project.org/package=wordcloud}