5 Text Mining

Copia e incolla da Pesce, con alcune integrazioni
ASSENTE 22 E 23 MAGGIO
Link video automi a stati finiti e Link video Automi a stati finiti transizioni di stato.

Siamo giunti ad analizzare dei dati che sono il meno strutturati possibile.
Se i nostri dati sono del testo, sono molto poco strutturati e quindi possiamo cercare delle regolarità utilizzando le espressioni regolari, che codificano in modo conciso insiemi finiti di stringhe, oppure contare quante parole ci sono nei documenti.

Abbiamo un corpus di documenti (insieme di documenti) e possiamo verificare quali sono le parole più frequenti all'interno di ciascun documento.

Per usare le espressioni regolari in R possiamo usare il pacchetto `stringr`. Ci sono dei caratteri speciali:

- . corrisponde a un qualsiasi carattere, tranne una nuova linea
- /d corrisponde ad un qualsiasi numero
- /s corrisponde ad uno spazio vuoto (spazio, tab, newline)
- [abc] per creare un insieme di caratteri che corrispondono ad un'enumerazione, cioè l'insieme dei caratteri a, b o c.
- [^abc] corrisponde a tutto tranne a, b o c.
- + corrisponde a 1 o più, chiusura positiva
- ? corrisponde a 0 o più occorrenze
- | corrisponde alla operazione di unione
- * corrisponde all'operazione di chiusura

Quindi, partendo dai simboli di un alfabeto, è possibile costruire un'espressione regolare e combinare i simboli con i caratteri speciali e con le operazioni.

Ci sono dei tools che permettono di fare espressioni regolari, in particolare:

1. Determinare quali stringhe hanno un match rispetto all'espressione regolare, cioè determinare quali stringhe appartengono ad un linguaggio riconosciuto da quell'espressione regolare
2. Trovare le posizioni che corrispondono ai match
3. Estrarre il contenuto dei match
4. Rimpiazzare la stringa con un nuovo valore
5. Dividere la stringa in base al match

Per determinare se un vettore di caratteri corrisponde ad un pattern, utilizzare `str_detect()`. Restituisce un vettore logico della stessa lunghezza dell'input:

```
x <- c("apple", "banana", "pear")
str_detect(x, "e") #sto chiedendo se la stringa "e" è contenuta nelle stringe
di x
#> [1]  TRUE FALSE  TRUE
```

Una variazione di `str_detect()` è `str_count()` che dice quanti match ci sono in una stringa:

```
x <- c("apple", "banana", "pear")
```

```r
str_count(x, "a")
#> [1] 1 3 1
```

Queste funzioni possono essere usate anche su vettori e l'output sarà un vettore.

Per estrarre il testo effettivo di una corrispondenza, utilizzare `str_extract()`, per una sola corrispondenza, `str_extract_all()` per estrarre tutte le corrispondenze. `str_replace()` e `str_replace_all()` permettono di sostituire in una stringa dei pezzi riconosciuti dall'espressione regolare con qualcosa

```r
x <- c("apple", "pear", "banana")
str_replace(x, "[aeiou]", "-")
#> [1] "-pple"  "p-ar"   "b-nana"
str_replace_all(x, "[aeiou]", "-")
#> [1] "-ppl-"  "p--r"   "b-n-n-"
```

`str_split()` suddivide una stringa rispetto ad un'espressione regolare.

Altra cosa che si può fare con un testo è contare la frequenza delle parole all'interno di un documento. In base al numero di parole di un certo tipo, si cerca di inferire il significato del testo, l'argomento. Possiamo anche vedere il sentimento che aveva lo scrittore quando scriveva quel testo, attraverso la **Sentiment Analysis**.

Se abbiamo un **corpus** di documenti, ossia una collezione di documenti, è possibile vedere per ogni documento, quali sono le parole più importanti, non solo in base alla frequenza delle parole, ma anche in base ad una tecnica chiamata **tf-idf** (temp fre frequency- inverse document frequency*)* che cerca di fare emergere le parole che sono caratterizzanti per quel documento e non per tutti i documenti.

Possiamo anche cercare di analizzare, non solo le singole parole, ma sequenze di parole accostate, gli **n-grammi**. Si crea una relazione tra più parole dettata dalla prossimità o consecutività delle parole stesse, ad esempio, se la parola "amore" è sempre seguita da "odio", oppure se sono presenti nello stesso paragrafo, allora nel documento vi è una relazione tra le due parole.

Infine, usando la tecnica **topic modelling**, proveremo a suddividere il corpus in cluster che corrispondono a comunità coese, cioè che trattano un certo topic, in modo fuzzy.

The tidy text format

È un approccio "tidy" perché cerca di sfruttare la forma normale, sostanzialmente dei dataframe dove le righe corrispondono alle osservazioni e le colonne alle variabili. Ma come è possibile inserire un testo all'interno di un dataframe? Fondamentalmente, il text mining si basa sul conteggio delle parole che vengono ripetute in un documento. Una volta contato il numero di parole, si può costruire un dataframe con tre colonne:

1. Prima colonna con informazioni sul documento
2. Seconda colonna informazioni sulla parola
3. Terza colonna indicazione sulla frequenza della parola nel documento

La prima cosa è fare un'operazione chiamata **tokenization**, leggere il testo, suddividerlo in token e per ogni token estratto incrementare la frequenza assoluta del token. Normalmente, un token è una parola (stringa separata da spazi). Successivamente è possibile usare `dplyr` o `ggplot` per fare delle analisi e delle visualizzazioni.

I pacchetti che useremo sono `tidyverse` e `tidytext`.

Sentiment Analysis

5.1 Espressioni regolari e automi

Iniziamo con una trattazione più teorica dalle dispense di Dovier e Giacobazzi ai capitoli 3 e 4 che si trovano sul sito.

5.1.1 Automi a stati finiti

Vogliamo studiare insiemi, detti *linguaggi*, costituiti da sequenze finite di caratteri presi da un dato insieme finito di simboli.
L'obiettivo è quello di introdurre due formalismi per descrivere insiemi infiniti di parole. Una parola è una sequenza di caratteri presi da un certo alfabeto. Definiamo alfabeto un insieme di simboli. Vorremmo caratterizzante in modo finito insiemi infiniti di parole. Ad esempio, tutte le stringhe che corrispondono a numeri divisibili per 2. Un alfabeto potrebbe essere {0,1} e 01101 è una parola.

- **Linguaggio**: insieme di parole su un alfabeto
- **Simbolo** è un elemento del nostro alfabeto
- **Stringa** (o parola) è una sequenza di caratteri (sequenza finita di simboli)
- **Lunghezza** di una parola è il numero di simboli, e si indica con |w|

Se prendessimo i numeri divisibili per 5, essi sono infiniti e quindi vogliamo dare una caratterizzazione finita. Per alcuni linguaggi, chiamati ***linguaggi regolari***, è possibile dare una caratterizzazione finita in termini di automi e espressioni regolari. Non sempre è possibile trovare una espressione regolare per un linguaggio.

Conviene fissare a questo punto un po' di sintassi.

Un *simbolo* è un elemento del nostro alfabeto, una *stringa* (o *parola*) è una sequenza di simboli giustapposti. Ad esempio, se a,b,c sono simboli, abcba è una stringa. Esiste una parola speciale ε, che è la stringa vuota, non contiene simboli, cioè $|\varepsilon| = 0$.

Sia w = a1 ⋯ an una stringa. Ogni stringa della forma:

- $a_1 \cdots a_j$, con $j \in \{1,\ldots,n\}$ è detta un prefisso di w
- $a_i \cdots a_n$, con $i \in \{1,\ldots,n\}$ è detta un suffisso di w
- $a_i \cdots a_j$, con $i,j \in \{1,\ldots,n\}$, $i \leq j$, è detta una sottostringa di w
- ε è sia prefisso che suffisso che sottostringa di w

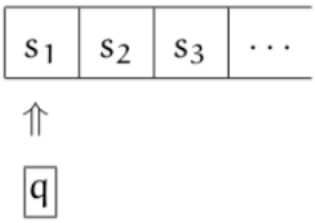
Osserviamo che ϵ è sia prefisso che suffisso che sottostringa di w, perché se n=0 allora w= ϵ. Ad esempio, i prefissi di abc sono ε, a, ab, e abc. I suffissi sono ε, c, bc, e abc. Le sottostringhe sono: ε, a ab abc, b bc

Una operazione importante è la concatenazione di due stringhe, se abbiamo due stringhe v e w, possiamo concatenarle e creare una nuova stringa z, z=vw. Si noti che (ab)c = a(bc).

Un **alfabeto** Σ è un insieme finito di simboli. Un **linguaggio** formale (in breve linguaggio) è un insieme di parole formate su un determinato alfabeto. L'insieme vuoto Ø e l'insieme {ε} sono due linguaggi formali di

qualunque alfabeto. Un linguaggio potrebbe essere finito, ma è un caso poco interessante perché ci interessa trovare caratteri finiti in casi infiniti. Con Σ*verrà denotato il linguaggio costituito da tutte le stringhe su un fissato alfabeto Σ. Dunque $\Sigma^*=\{ a_1, \ldots, a_n : n \geq 0, a_n \in \Sigma\}$. Ad esempio, se Σ = {0}, allora Σ* = {ε,0,00,000,...}. Se Σ = {0,1}, allora Σ*={ε,0,1,00,01,10,11,000,...} cioè tutte le parole che posso scrivere con questi due simboli.

AUTOMA: Un **automa** è fondamentalmente un modello di calcolo a stati finiti, cioè si può trovare inun numero finito di stati. Un esempio è l'ascensore: se una persona è allo stato 3, l'automa leggerà la sequenza di piani che vengono pigiati. Si può rappresentare mediante una come una macchina che ha un registro di memoria che contiene lo stato q e legge una sequenza di simboli, cioè una parola finita.

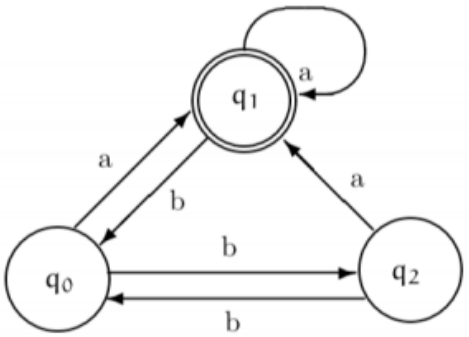


A partire dallo stato in cui si trova l'automa legge un simbolo e cambia eventualmente lo stato, poi la tesina si sposta di uno in avanti, cioè legge il simbolo successivo (l'ascensore sale di uno). Dobbiamo immaginarlo come una sorta di macchina che ha un unico registro di memoria e in base allo stato in cui si trova e in base al simbolo che trova decide di andare in un altro stato. É importante notare che l'automa può spostarsi in un numero finito di simboli e può leggere solo un simbolo alla volta.

Il movimento dell'automa è rappresentabile tramite un **grafo** o una **matrice di transizione**, che non ha nulla a che fare con la matrice di adiacenza. La matrice di transizione ha sulle righe gli stati e sulle colonne i simboli. Ad esempio, qui abbiamo 3 stati e 2 simboli, ci specifica il comportamento dell'automa:

	a	b
q_0	q_1	q_2
q_1	q_1	q_0
q_2	q_1	q_0

Ci dice che sono nello stato q_0 e leggo il simbolo a allora nel passo successivo andrò nello stato q_1.



I nodi sono gli stati e gli archi sono i simboli letti. Quindi il comportamento dell'automa può essere identificato in questi due modi. I nodi cerchiati due volte sono quelli finali. i.

La differenza con il formalismo della macchina di Turing, che è un marchingegno che legge dei simboli su un nastro e a partire dagli stati produce degli output, sono:

1. L'automa a stati finiti non crea output (può solo spostarsi di stato in stato)
2. L'automa può andare solo avanti e non può tornare indietro, non può rivedere quale sia la sua storia. Tutta la storia viene codificata nello stato attuale

Se prendiamo gli automi e aggiungiamo queste due proprietà otteniamo il **formalismo** più espressivo che si chiama **Macchina di Turing**. Ha un formalismo molto importante perché rappresenta il formalismo di ogni calcolatore di qualsiasi grandezza. Possiamo immaginar allora il nostro computer come una testina che ha un output e può andar avanti e indietro. Proviamo a dare una definizione più formale.

5.1.2 Automi deterministici

Un **automa a stati finiti deterministico** (**DFA**) è una quintupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ dove:

- Q è un insieme finito di stati;
- Σ è un alfabeto (alfabeto di input);
- $\delta: Q \times \Sigma \rightarrow Q$ è la funzione di transizione;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali.

Quindi per ogni stato e per ogni simbolo l'automa deve sapere che cosa fare; per ogni nodo ci deve essere un numero di archi pari al numero di simboli che escono da quel nodo. Gli stati finali corrispondono agli stati che servono per decidere il linguaggio che l'automa riconosce.

Come definire il linguaggio accettato dal nostro automa? Intuitivamente faccio così: dobbiamo decidere se una parola appartiene al linguaggio, con l'automa leggo la parola e se arrivo allo stato finale allora la parola è accettata, altrimenti viene rifiutata.

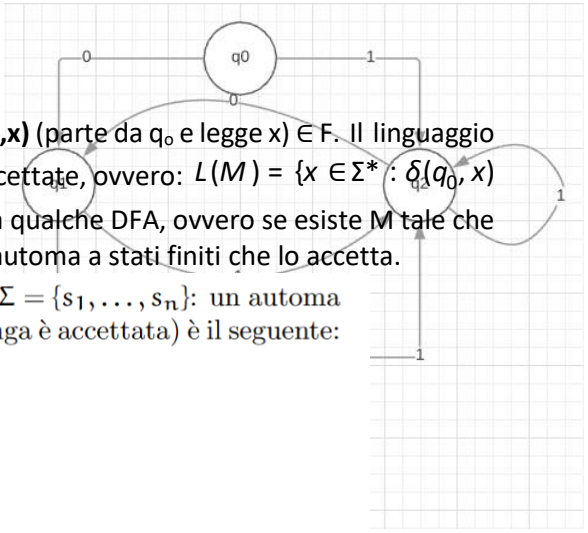
Useremo:

- p, q, r con o senza pedici per denotare stati,
- P, Q, R, S per insiemi di stati,

- a, b con o senza pedici per denotare simboli di Σ,
- x, y, z, u, v, w con o senza pedici per denotare stringhe.

Dalla funzione δ si ottiene in modo univoco la funzione $\delta : Q x \Sigma^* : -> Q$ nel modo seguente:

$\delta^{\wedge}(q,\varepsilon) = q$
$\delta^{\wedge}(q,wa) = \delta(\delta^{\wedge}(q,w), a)$ (dove w è una parola e a è un simbolo)



Una stringa x è detta **accettata** da un DFA M= {Q,Σ,δ,q_o,F} se **δ^ (q_o,x)** (parte da q_o e legge x) ∈ F. Il linguaggio accettato da M, denotato come L(M) è l'insieme delle stringhe accettate, ovvero: $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \in F\}$. Un linguaggio L è detto **linguaggio regolare** se è accettato da qualche DFA, ovvero se esiste M tale che L = L(M). Quindi un linguaggio è detto regolare se esiste qualche automa a stati finiti che lo accetta.

ESEMPIO 3.5. $\varnothing$ e Σ^* sono linguaggi regolari. Sia $\Sigma = \{s_1, \ldots, s_n\}$: un automa M_0 che riconosce il linguaggio $\varnothing$ (ovvero: nessuna stringa è accettata) è il seguente:

	s_1	...	s_n
q_0	q_0	...	q_0

ove $F = \varnothing$. Infatti, poiché $\forall x\,(x \notin \varnothing)$, si ha che:

$$(\forall x \in \Sigma^*)(\hat{\delta}(q_0, x) \notin F)\,.$$

Un automa per Σ^*, è invece l'automa M_1:

	s_1	...	s_n
q_0	q_0	...	q_0

ove $F = \{q_0\}$. Si dimostra facilmente infatti, per induzione su $|x|$ che

$$(\forall x \in \Sigma^*)(\hat{\delta}(q_0, x) = q_0)\,.$$

Facciamo alcuni semplici esempi.

Esempio: proviamo a creare un automa che riconosce il linguaggio vuoto, (qualsiasi parola gli diamo non gli va bene) che sicuramente è un linguaggio perché sotto insieme di Σ*, allora basterà mettere l'insieme degli stati finali uguale a vuoto: F = ∅. Esiste un unico stato q_o e, qualsiasi simbolo l'automa leggerà, rimane sempre in quello stato. Ma q_o non è uno stato finale in quanto l'insieme degli stati finali è vuoto. Questo automa è quello **scettico** a cui non va bene niente.

Esempio: automa a cui va bene tutto. L'insieme degli stati finali è q_o, qualsiasi parola leggerà andrà sempre nello stato q_o ma questo è uno stato finale. È l'automa a cui gli va bene tutto, il cosiddetto **credulone**, è speculare al primo.

Esempio: scrivere l'automa che riconosce tutti e soli i numeri pari, ovvero tutte le stringhe nell'insieme Σ = {0,1} tali che, se interpretate come numero intero (binario), sono divisibili per 2.

In realtà per scrivere un automa basta rappresentare il grafo delle transazioni.

- Stato q_0= stato in cui leggo un numero pari (è lo stato finale)
- Stato q_1= stato in cui leggo un numero dispari (numero che finisce per 1)

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

Proviamo con la parola 101 (leggo 1, leggo 0, leggo 1) sono in uno stato non finale quindi la parola non viene accettata. La stringa vuota viene accettata. Invece 100 viene accettata.

Esempio: automa che riconosce i numeri divisibili per 4 (quindi 4,8,12 ecc), che sono quelli che terminano, in binario, con due zeri (quindi ad esempio la stringa 100,1000). Ogni stato rappresenta un possibile resto della divisione per 4 (posso avere 0,1,2,3 ecc).

- Stato q0 = il numero che ho letto fino ad ora, diviso per 4, ha resto 0 (che è lo stato finale)
- Stato q1 = il numero che ho letto fino ad ora, diviso per 4, ha resto 1

- Stato q2 = il numero che ho letto fino ad ora, diviso per 4, ha resto 2
- Stato q3 = il numero che ho letto fino ad ora, diviso per 4, ha resto 3

Se leggo 0 è sicuramente divisibile per 4, quindi rimarrò lì. Se invece leggo 1 (1/4 fa zero con il resto di uno e quindi vado nello stato 1). Se leggo 10 è 2 e la divisione per 4 fa 0 con il resto di 2 e andrà nello stato 2. Leggo 11 è 3 e la divisione per 4 fa o con il resto di 3, quindi vado nello stato q_3. 100 è 4, quindi devo tornare il q_0. 101 cioè 5 che diviso per 4 fa uno con il resto di 1 quindi vado in q_1. 110 cioè 6 che diviso per 4 fa 0 con il resto di 2 quindi devo andare in q_2. Infine 111 cioè 7 che diviso per 4 fa 1 con il resto di 3, quindi rimango in q_3. Ora il mio automa è completo.

	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

Automi deterministici: in qualunque stato mi trovo poi al massimo andrò in un unico altro stato (ma non può andare in più stati o andare in nessun stato). Lo posso vedere come un cammino in cui non ho possibilità di scelta.

5.1.3 Automi non deterministici

Un altro formalismo è basta sugli automi non deterministici. Automa non deterministico: è un automa che può percorrere più strade parallelamente. Se si trova in uno stato e legge un simbolo può andare in due o tre o più stati diversi, oppure in nessuno.

Un **automa a stati finiti non-deterministico** (**NFA**) è una quintupla $\langle Q, \Sigma, \delta, q0, F\rangle$ dove

- Q,
- Σ,
- q_0 e
- $F \subseteq Q$
- mantengono il significato visto per gli automi deterministici, mentre ora devo cambiare la funzione di transizione δ che è definita

$$\delta : Q \times \Sigma \rightarrow \wp(Q)$$

Si osservi che è ora ammesso: δ(q,a) = ∅ per qualche q ∈ Q ed a ∈ Σ. Anche per gli NFA dalla funzione δ si ottiene in modo univoco la funzione δ^: Q×Σ∗ → ℘(Q) (insieme delle parti di Q) nel modo seguente:

Si osservi che è ora ammesso: δ(q,a) = ∅ per qualche q ∈ Q ed a ∈ Σ. Anche per gli NFA dalla funzione δ si ottiene in modo univoco la funzione δ: Q×Σ∗ → ℘(Q) nel modo seguente:

$$\delta(q, \epsilon) = \{q\}$$

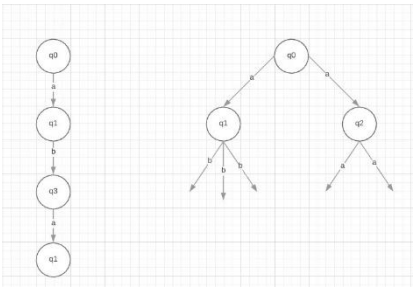
$$\delta(q, wa) = \bigcup_{p \ in \delta(q,w)} \delta(p, a)$$ (Unione $p \in \delta\hat{}(q,w)$ δ(p,a) lo stato che mi trovo dopo aver letto una parola è l'unione degli stati possibili, devo seguire tutte le strad)

Una stringa x è accettata da un NFA M = ⟨Q, Σ, δ, q0, F⟩ se δ(q0, x) ∩ F≠∅. Il linguaggio accettato da M è l'insieme delle stringhe accettate, ovvero: L(M)={x∈Σ* :δ(q_0,x)∩F≠∅}.

Una stringa x è accettata da un NFA M= {Q,Σ,δ,q_0,F} se **δ^ (q_0,x) ∩ F diverso ∅** (esiste almeno uno stato che ho raggiunto che non è vuoto).

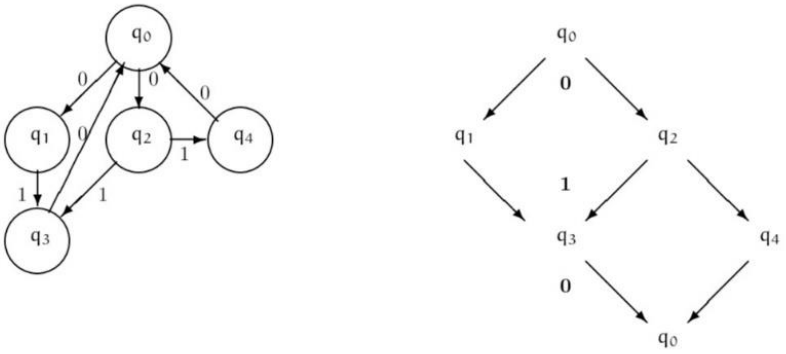
Il linguaggio accettato da M è l'insieme delle stringhe accettate, ovvero: L(M) = {x ∈ Σ^* :δ ^ (q_0,x) ∩ F diverso ∅}. Quindi data una parola il mio automa risponderà con un insieme di stati, può essere vuoto o contenere uno o più stati.

A questo puto data una parola il mio automa non risponderà più con un unico stato ma con più stati. Per vedere se la parola è accetta, se almeno una delle computazioni è andata a buon fine. Deve esistere almeno uno stato che ho raggiunto che non è vuoto.



Un automa deterministico, semplificando, può esser visto come un cammino. Nel caso non deterministico abbiamo un albero, come in figura.

Apparentemente questo è un formalismo molto più potente. In verità non lo è. Perché è possibile dimostrare un teorema di equivalenza tra i due formalismi. i linguaggi accettati da un automa deterministico e non deterministico sono gli stessi, sono i linguaggi regolari; quindi, per ogni automa non deterministico, è possibile costruire un automa deterministico che accetta lo stesso linguaggio (il viceversa è ovvio siccome l'automa deterministico è un caso particolare di uno non deterministico).



Prima di vedere il teorema inverso, ragioniamo sull'esempio in figura qui sopra. Proviamo a seguire la computazione dell'automa sulla stringa 010. All'inizio la computazione si biforca in modo non deterministico sui due stati q1 e q_2. Ciò può erroneamente far pensare che sia necessaria una struttura dati ad albero per rappresentare una computazione non-deterministica. Al secondo livello, quando il carattere 1 è analizzato, sia da q_1 che da q_2 si raggiunge lo stato q_3. Non è necessario ripetere lo stato in due nodi distinti. Inoltre lo stato q_4 è pure raggiungibile da q_2. Proseguendo, si vede che le due computazioni non deterministiche confluiscono nello stato q_0.

Il passaggio da automa non deterministico ad automa deterministico può determinare una esplosione esponenziale del numero degli stati.

5.1.4 Automi-transizioni

L'ultimo formalismo sono gli **automi con ε-transizioni**. In questo caso l'idea è che l'automa può cambiare lo stato anche se non legge alcun simbolo (carattere in input), può muoversi di stato in stato. Un NFA con ε-transizioni è una quintupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ dove:

- Q,
- Σ
- q_0 e
- $F \subseteq Q$
- sono come per gli automi non deterministici, mentre la funzione di transizione δ è ora definita

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \wp(Q) .$$

ogni automa con ε transazioni corrisponde ad un automa non deterministico e quindi anche ad un automa deterministico.

La costruzione della funzione $\delta : Q \times \Sigma^* \rightarrow \wp(Q)$ nel caso dei ε-NFA risulta leggermente più complessa che nei casi precedenti. Per far ciò si introduce la **funzione *ε-closure*** che, applicata ad uno stato, restituisce l'insieme degli stati raggiungibili da esso (compreso sé stesso) mediante ε-transizioni. La costruzione di tale funzione è equivalente a quella che permette di conoscere i nodi raggiungibili da un nodo in un grafo e può facilmente essere calcolata a partire dalla funzione δ (un arco $p \rightarrow q$ si ha quando $q \in \delta(p, \varepsilon)$). Il concetto di ε-closure si estende in modo intuitivo ad insiemi di stati:

$$\epsilon\text{-}closure(P) = \bigcup_{p \in P} \epsilon\text{-}closure(p)$$

$\hat{\delta}$ si può ora definire nel modo seguente:

$$\begin{cases} \partial(q, \epsilon) = \epsilon\text{-}closure(q) \\ \hat{\delta}(q, wa) = \bigcup_{p\ in \hat{\delta}(q,w)}\ \epsilon\text{-}closure(delta(p, a)) \end{cases}$$

Si noti che in questo caso $\hat{\delta}(q,a)$ può essere diverso da $\delta(q,a)$. Ad esempio, nell'automa:

Si ha che $\delta(q,a) = \{q'\}$, mentre $\hat{\delta}(q,a) = \bigcup_{p \text{ in } \delta(q,w)} \epsilon\text{-}closure(delta(p, a)) = \{q', q''\}$.

Definiamo dunque il linguaggio accettato dall'automa $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \cap F \neq \emptyset\}$.

Definiamo dunque il linguaggio accettato dall'automa $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \cap F \neq \emptyset\}$.
Si osservi come per questa classe di linguaggi si potrebbe assumere che l'insieme F abbia esattamente un elemento. Si osservi inoltre che $\hat{\delta}(q, x) = \varepsilon\text{-closure}(\delta(q, x))$.

Anche questa cosa non aggiunge espressività, esso è equivalente ad un automa non deterministico e quindi anche un automa deterministico.

5.1.5 Espressioni Regolari

Sono espressioni in un'algebra elementi insiemi di parole su un alfabeto con le operazioni di unione, concatenazione e chiusura su insiemi.

Una Espressione Regolare (ER) è una espressione in una algebra dove gli *elementi* sono insieme di parole su un alfabeto e le *operazioni* sono operazioni su insiemi: unione, concatenazione e chiusura. Ogni volta che applico una operazione ho un elemento del mio dominio.

- La concatenazione di L1 e L2, denotata come L1L2 è l'insieme: $L1L2 = \{xy \in \Sigma^* : x \in L1, y \in L2\}$.
- Unione: L1 + L2
- La chiusura (di Kleene) di L, denotata come L^* è l'insieme: $L* = \text{Unione}_{i \geq 0}\ L^i$ dove L_i è l'i-esima concatenazione.

Ad esempio, per quanto riguarda la chiusura: (0 + 1) * con i simboli 0 e 1 con la parola vuota. Sono tutte le parole che posso scrivere.

Def. Formale. Sia Σ un alfabeto. Espressione regolare su Σ: insieme che può essere ottenuto ricorsivamente in questo modo:

- ∅ è una espressione regolare che denota l'insieme vuoto
- ε è una espressione regolare che denota l'insieme {ε}
- Per ogni simbolo a ∈ Σ, a è una espressione regolare che denota l'insieme {a} (insieme che contiene solo quel simbolo)
- Se r e s sono espressioni regolari denotanti rispettivamente gli insiemi R ed S, allora (r+s), (rs), e (r∗) sono espressioni regolari che denotano gli insiemi R∪S, RS, e R∗ rispettivamente

Se r è una espressione regolare, indicheremo con L(r) il linguaggio denotato da r.

Tra espressioni regolari valgono delle uguaglianze che permettono la loro manipolazione algebrica. Varrà che r = s se e solo se L(r) = L(s).

Esempio: scriviamo le espressioni regolari degli automi di prima. Espressione regolare che identifica i numeri pari. L'alfabeto è {0,1}.

r = (0+1)* + 0 (quindi all'inizio riconosco qualsiasi cosa dell'alfabeto ma l'importante è che finisca con uno zero.

Esempio: espressione regolare che identifica i numeri divisibili per 4. L'alfabeto è {0,1}.

r = (0+1)* + 00

Non sempre le espressioni sono più semplici degli automi!!

TEOREMA DI EQUIVALENZA TRA DFA e ER → espressioni regolari ed automi hanno la stessa espressività, quindi per ogni automa esiste una corrispondente espressione regolare e viceversa.

- Sia r una espressione regolare. Allora esiste un ε-NFA M tale che L(M) = L(r)
- Sia M un DFA. Allora esiste una espressione regolare r tale che L(M) = L(r). Per la dimostrazione mi devo chiedere come arrivo a quello stato (es. arrivo in q_0 da q_2 leggendo 1) e risolvo il sistema di equazioni.

(Vedi dimostrazione su dispensa).

Vediamo ora le espressioni regolari in R. Se i nostri dati sono del testo possiamo cercare delle espressioni regolari oppure possiamo contare quante parole ci sono nei nostri documenti (o vedere quali sono le parole più frequenti). The stringr package for string manipulation. Stringr is not part of the core tidyverse because you don't always have textual data, so we need to load it explicitly. Caratteri speciali: . che corrisponde a qualsiasi carattere, \v che corrisponde a qualsiasi numero, \s che corrisponde a qualsiasi spazio, se vogliamo creare un insieme di caratteri abc possiamo farlo con le parentesi [ ], oppure se aggiungiamo un ^ prima dell'insieme vuol dire prenderà tutto tranne quei caratteri. Operazioni: unione, si indica con |

- ?: 0 or 1
- +: 1 or more (chiusura positiva)
- *: 0 or more (chiusura)

Quindi possiamo scrivere un'espressione regolare partendo dai simboli del nostro alfabeto (es. lettere) e poi combinarli con i caratteri speciali e con le 4 operazioni.

Ci sono dei tools che ci permettono di usare le espressioni regolari, possiamo fare 5 operazioni:

- Determinare quali stringhe appartengono al linguaggio riconosciuto da quell'espressione regolare
- Trovare le posizioni che corrispondono ai match
- Estrarre il contenuto dei match
- Rimpiazzare i match con altro
- Suddividere la stringa in base ai match

str_extract(), str_replace(), str_split()

Sommario prossimi argomenti

Altra cosa che possiamo fare quando abbiamo un testo: contare le parole o la frequenza delle parole in modo da capire di che cosa sta parlando il testo. Possiamo anche vedere il sentimento che aveva lo scrittore quando scriveva quel testo (sentiment analysis), emozioni che emergono da quel testo. Se abbiamo una collezione di documenti (es. un certo numero di romanzi) possiamo vedere per ogni documento quali sono le parole più importanti, utilizzando la tecnica tf-idf (cerca di catturare le parole che caratterizzano quel documento). Possiamo analizzare non solo le singole parole, ma anche i cosiddetti n-grams, sequenze di parole accostate (legame tra nomi, relazioni), due parole adiacenti oppure due parole che sono nella stessa sezione (parole collegate). Infine vedremo, utilizzando la tecnica topic modelling, come suddividere il nostro insieme di documenti in cluster che parlano di un certo topic (es. il nostro documento parla 20% di finanza e 80% di politica), i topic non verranno scelti a priori.

THE TIDY TEXT FORMAT. Questo approccio si chiama tidy perché cerca di sfruttare la forma normale che abbiamo visto, anche per analizzare il testo. Dovremo usare i dataframe nel senso che ad esempio la prima colonna del documento mi parla dell'articolo 517, la seconda colonna è parola che si trova in quell'articolo e la terza colonna mi dice la frequenza della parola in quell'articolo. Usiamo i dataframe non per rappresentare il testo ma per fare una sorta di summary. L'input è un corpus (collezione di documenti), un documento è un pezzo di testo NON strutturato. Possiamo fare un'operazione di **tokenization** (processo che suddivide un testo in token), ovvero ogni token (pezzo, stringa separata da spazi) che incontriamo lo estraiamo e incrementiamo la frequenza assoluta di quel testo. Poi possiamo usare dplyr o ggplot per fare delle analisi.

I due pacchetti che useremo sono tidyverse e tidytext.

Si vede ora brevemente le espressioni regolari in R capitolo 14 di R for DataScience(R 4 Data Science).

5.2. Text Mining with R - Un approccio ordinato:

2 Analisi del sentimento con dati ordinati

Nel capitolo precedente, abbiamo esplorato in modo approfondito cosa intendiamo per il formato del testo ordinato e abbiamo mostrato come questo formato può essere utilizzato per affrontare le domande sulla frequenza delle parole. Questo ci ha permesso di analizzare quali parole sono usate più frequentemente nei documenti e di confrontare i documenti, ma ora esaminiamo un argomento diverso. Affrontiamo il tema

dell'analisi di opinione o analisi del sentimento. Quando noi lettori umani si avvicinano a un testo, utilizziamo la nostra comprensione dell'intento emotivo delle parole per dedurre se una parte del testo è positiva o negativa, o forse caratterizzata da altre emozioni più sfumate come la sorpresa o il disgusto. Possiamo utilizzare gli strumenti di text mining per accedere al contenuto emotivo del testo a livello di programmazione, come mostrato nella Figura 2.1 .

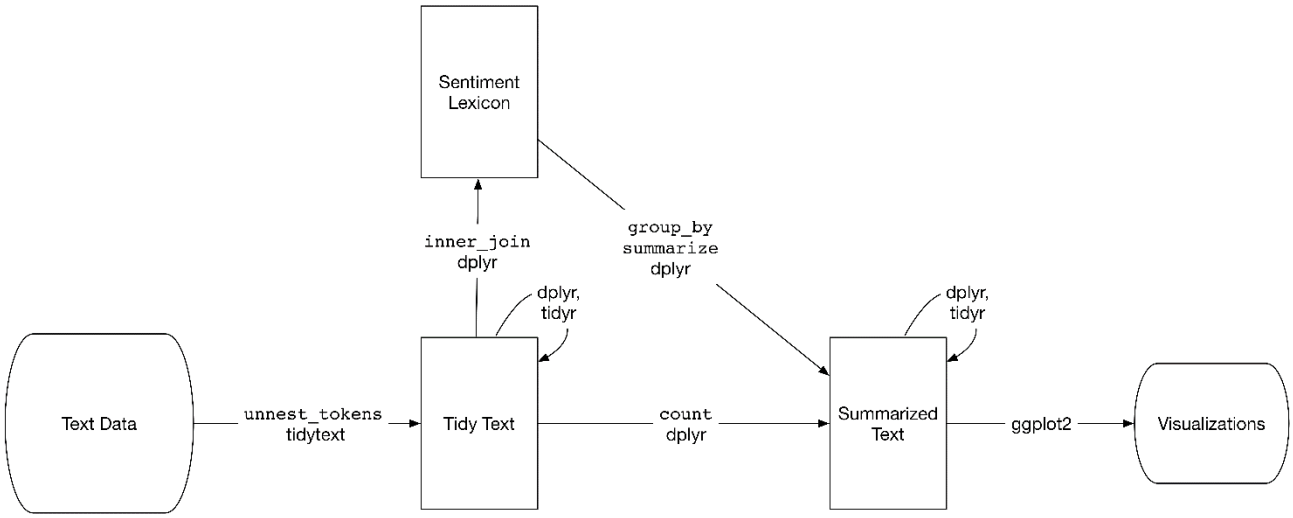


Figura 2.1: Un diagramma di flusso di un'analisi di testo tipica che utilizza il testo tidy per l'analisi del sentiment. Questo capitolo mostra come implementare l'analisi delle opinioni utilizzando i principi dei tidy data.

Un modo per analizzare il sentimento di un testo è considerare il testo come una combinazione delle sue singole parole e il contenuto di sentimento dell'intero testo come la somma del contenuto di sentimento delle singole parole. Questo non è l'unico modo per affrontare l'analisi dei sentimenti, ma è un approccio spesso utilizzato e un approccio che sfrutta naturalmente l'ecosistema degli strumenti ordinato.6

5.2.1 Il sentiments set di dati

Come discusso sopra, ci sono una varietà di metodi e dizionari che esistono per valutare l'opinione o l'emozione nel testo. Il pacchetto tidytext contiene diversi lessici di sentimento nel `sentiments` set di dati.

```
library(tidytext)

sentiments
## # A tibble: 27,314 x 4
##    word        sentiment lexicon score
##    <chr>       <chr>     <chr>   <int>
##  1 abacus      trust     nrc        NA
##  2 abandon     fear      nrc        NA
##  3 abandon     negative  nrc        NA
##  4 abandon     sadness   nrc        NA
##  5 abandoned   anger     nrc        NA
##  6 abandoned   fear      nrc        NA
##  7 abandoned   negative  nrc        NA
##  8 abandoned   sadness   nrc        NA
##  9 abandonment anger     nrc        NA
## 10 abandonment fear      nrc        NA
## # ... with 27,304 more rows
```

I tre lessici generici sono
• AFINN da Finn Årup Nielsen ,
• bingda Bing Liu e collaboratori , e
• Saif Mohammad e Peter Turney .

Tutti e tre questi lessici sono basati su **unigram**, cioè parole singole. Questi lessici contengono molte parole inglesi e alle parole vengono assegnati punteggi per il sentimento positivo / negativo, e possibilmente anche emozioni come gioia, rabbia, tristezza e così via.

- Il lessico nrc categorizza le parole in modo binario ("sì" / "no") in categorie di positivo, negativo, rabbia, anticipazione, disgusto, paura, gioia, tristezza, sorpresa e fiducia.
- Il lessico bing classifica le parole in modo binario in categorie positive e negative.
- Il lessico AFINN assegna parole con un punteggio compreso tra -5 e 5, con punteggi negativi che indicano un sentimento negativo e punteggi positivi che indicano un sentimento positivo. Tutte queste informazioni sono catalogate nel sentiments set di dati e tidytext fornisce una funzione get_sentiments() per ottenere specifici lessici del sentimento senza le colonne che non sono utilizzate in quel lessico.

```
get_sentiments("afinn")
## # A tibble: 2,476 x 2
##    word       score
##    <chr>      <int>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # ... with 2,466 more rows
```

```
get_sentiments("bing")
## # A tibble: 6,788 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faced     negative
##  2 2-faces     negative
##  3 a+          positive
##  4 abnormal    negative
##  5 abolish     negative
##  6 abominable  negative
##  7 abominably  negative
##  8 abominate   negative
##  9 abomination negative
## 10 abort       negative
## # ... with 6,778 more rows
```

```
et_sentiments("nrc")
```

```
## # A tibble: 13,901 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

In che modo questi lessici del sentimento sono stati messi insieme e convalidati? Sono stati costruiti tramite crowdsourcing (usando, ad esempio, Amazon Mechanical Turk) o con il lavoro di uno degli autori, e sono stati convalidati utilizzando nuovamente alcune combinazioni di crowdsourcing, recensioni di ristoranti o film, o dati di Twitter. Date queste informazioni, potremmo esitare ad applicare questi lessici di sentimento a stili di testo notevolmente diversi da quelli su cui sono stati convalidati, come la narrativa narrativa di 200 anni fa. Se è vero che l'uso di questi lessici del sentimento con, per esempio, i romanzi di Jane Austen può darci risultati meno accurati rispetto ai tweet inviati da uno scrittore contemporaneo, possiamo ancora misurare il contenuto del sentimento per le parole che sono condivise attraverso il lessico e il testo.

Sono disponibili anche alcuni lessici sul sentimento specifici del dominio, costruiti per essere utilizzati con il testo di un'area di contenuto specifica. La Sezione 5.3.1 esplora un'analisi utilizzando un lessico di opinioni specifico per la finanza.

I metodi basati sul dizionario come quelli di cui stiamo discutendo trovano il sentimento totale di una parte di testo sommando i singoli punteggi di sentimento per ogni parola nel testo.

Non tutte le parole inglesi sono nei lessici perché molte parole inglesi sono piuttosto neutre. È importante tenere presente che questi metodi non tengono conto dei qualificatori prima di una parola, come "non buono" o "non vero"; un metodo basato sul lessico come questo è basato esclusivamente su unigram. Per molti tipi di testo (come gli esempi narrativi di seguito), non ci sono sezioni sostenute di sarcasmo o testo negato, quindi questo non è un effetto importante. Inoltre, possiamo usare un approccio ordinato per iniziare a capire quali tipi di parole di negazione sono importanti in un determinato testo; vedere il capitolo 9 per un esempio esteso di tale analisi.

Un'ultima avvertenza è che la dimensione del frammento di testo che usiamo per sommare i punteggi di sentimento unigramma può avere un effetto su un'analisi. Un testo delle dimensioni di molti paragrafi può spesso avere un sentimento positivo e negativo mediamente pari a circa zero, mentre il testo a dimensioni di frase o paragrafo funziona spesso meglio.

2.2 Analisi del sentimento con inner join

SENTIMENT ANALYSIS. Come facciamo ad analizzare il sentimento di un testo? Ci sono vari metodi, noi utilizzeremo quello basato sul sentimento, che afferma: il sentimento totale di un testo è la somma dei sentimenti delle singole parole del testo. Dovremo quindi etichettare

ogni parola con un sentimento. Per dare un sentimento alle parole esistono dei dataset (che contengono i sentimenti delle parole), i 3 più famosi sono:

- AFINN from Finn Årup Nielsen,
- bing from Bing Liu and collaborators, and
- nrc from Saif Mohammad and Peter Turney.

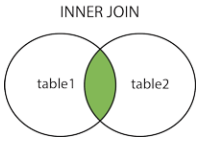
Sono contenuti nel pacchetto tidytext, per vederli usiamo ad esempio get_sentiments("afinn"), c'è una variabile score in cui numeri positivi corrispondono a sentimenti positivi e numeri negativi corrispondono a sentimenti negativi.

Dovremo stare attenti alle negazioni.

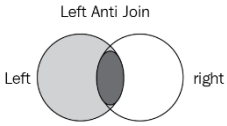
Abbiamo un lessico che per ogni parola ci dice qual è il sentimento, ora useremo il lessico "nrc" che include vari sentimenti, tra cui gioia. Andiamo a vedere nel nostro documento quante parole di questo tipo ci sono.

22 Maggio

Con i dati in un formato ordinato, l'analisi del sentiment può essere eseguita come join interno. Questo è un altro dei grandi successi della visualizzazione del text mining come un ordinato compito di analisi dei dati; tanto quanto la rimozione delle parole di arresto è un'operazione antijoin, l'esecuzione dell'analisi del sentiment è un'operazione di inner join.



Diamo un'occhiata alle parole con un punteggio di gioia dal lessico NRC. Quali sono le parole di gioia più comuni in Emma? Per prima cosa, dobbiamo prendere il testo dei romanzi e convertire il testo nel formato ordinato usando **unnest_tokens(),** proprio come abbiamo fatto nella Sezione 1.3 . Impostiamo anche alcune altre colonne per tenere traccia di quale linea e capitolo del libro ogni parola proviene; usiamo group_by e mutate costruiamo quelle colonne.




```
library(janeaustenr)
library(dplyr)
library(stringr)

tidy_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                 ignore_case = TRUE)))) %>%
  ungroup() %>%
  unnest_tokens(word, text)
```

Si noti che abbiamo scelto il nome word per la colonna di output da unnest_tokens(). Questa è una scelta conveniente perché i lessici di sentimento e i dataset di fine parola hanno colonne di nome word; eseguire join interni e anti-join è quindi più facile.

Ora che il testo è in un formato ordinato con una parola per riga, siamo pronti a fare l'analisi del sentimento. Per prima cosa, usiamo il lessico NRC e filter() per le parole di gioia. Quindi, passiamo filter() al frame

dei dati con il testo dei libri per le parole di Emma e quindi utilizziamo `inner_join()` per eseguire l'analisi dei sentimenti. Quali sono le parole di gioia più comuni in Emma? Usiamo `count()` da dplyr.

```
nrc_joy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

tidy_books %>%
  filter(book == "Emma") %>%
  inner_join(nrc_joy) %>%
  count(word, sort = TRUE)
## # A tibble: 303 x 2
##    word        n
##    <chr>   <int>
##  1 good      359
##  2 young     192
##  3 friend    166
##  4 hope      143
##  5 happy     125
##  6 love      117
##  7 deal       92
##  8 found      92
##  9 present    89
## 10 kind       82
## # ... with 293 more rows
```

Qui vediamo per lo più parole positive sulla speranza, l'amicizia e l'amore. Vediamo anche alcune parole che non possono essere usate con gioia da Austen ("trovato", "presente"); discuteremo di questo in maggior dettaglio nella Sezione 2.4.

Possiamo anche esaminare come cambiano le opinioni in ogni romanzo. Possiamo farlo con solo una manciata di linee che sono per lo più funzioni dplyr. In primo luogo, troviamo un punteggio sentiment per ogni parola usando il lessico di Bing e `inner_join()`.

Successivamente, contiamo quante parole positive e negative ci sono in sezioni definite di ogni libro. Definiamo un `index` qui per tenere traccia di dove siamo nella narrazione; questo indice (usando la divisione intera) conta sezioni di 80 righe di testo.

L' %/%operatore esegue la divisione intera (x %/% y è equivalente a floor(x/y)) in modo che l'indice tenga traccia di quale sezione di testo di 80 righe stiamo contando il sentimento negativo e positivo in.

Piccole sezioni di testo potrebbero non contenere abbastanza parole per ottenere una buona stima del sentimento, mentre sezioni molto ampie possono eliminare la struttura narrativa. Per questi libri, l'uso di 80 linee funziona bene, ma questo può variare a seconda dei singoli testi, per quanto tempo devono iniziare le righe, ecc. Quindi usiamo in `spread()` modo da avere un sentimento negativo e positivo in colonne separate e infine calcolare un sentimento netto (positivo - negativo).

```
library(tidyr)

jane_austen_sentiment <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

Ora possiamo tracciare questi punteggi sentimentali attraverso la traiettoria della trama di ogni romanzo. Si noti che stiamo tracciando i sentimenti contro l'asse x = indice che tiene traccia del tempo narrativo in sezioni di testo.

```
library(ggplot2)

ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```

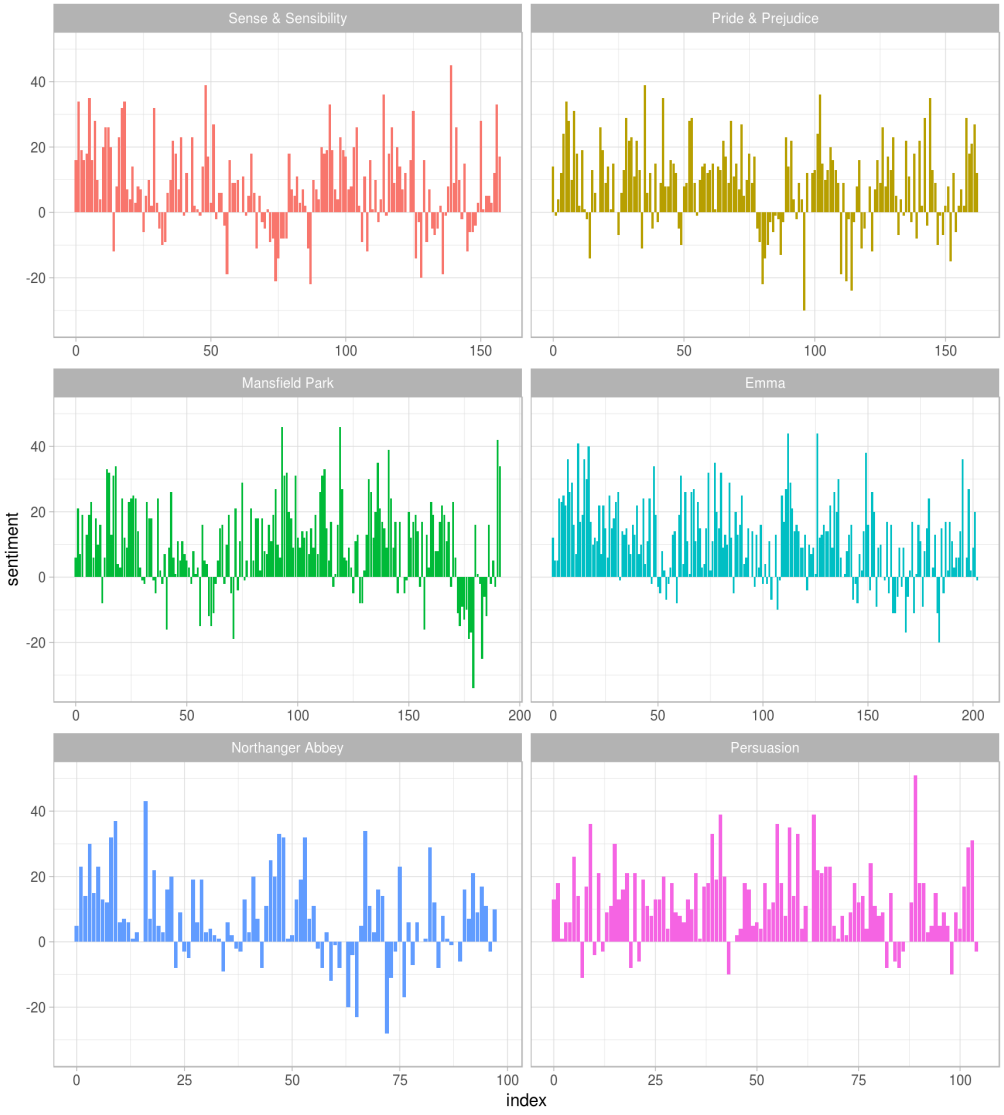


Figura 2.2: Sentimento attraverso le narrazioni dei romanzi di Jane Austen

Possiamo vedere in Figura 2.2 come la trama di ogni romanzo cambia verso un sentimento più positivo o negativo sulla traiettoria della storia.

5.2.3 Confronto tra i tre dizionari di sentiment

Con diverse opzioni per i lessici del sentiment, potresti volere qualche informazione in più su quale sia appropriato per i tuoi scopi. Usiamo tutti e tre i lessici del sentimento ed esaminiamo come il sentimento

cambia attraverso l'arco narrativo di *Pride and Prejudice*. Per prima cosa, usiamo `filter()` solo le parole del singolo romanzo a cui siamo interessati.

```
pride_prejudice <- tidy_books %>%
  filter(book == "Pride & Prejudice")

pride_prejudice
```

```
## # A tibble: 122,204 x 4
##    book               linenumber chapter word
##    <fct>                   <int>   <int> <chr>
##  1 Pride & Prejudice           1       0 pride
##  2 Pride & Prejudice           1       0 and
##  3 Pride & Prejudice           1       0 prejudice
##  4 Pride & Prejudice           3       0 by
##  5 Pride & Prejudice           3       0 jane
##  6 Pride & Prejudice           3       0 austen
##  7 Pride & Prejudice           7       1 chapter
##  8 Pride & Prejudice           7       1 1
##  9 Pride & Prejudice          10       1 it
## 10 Pride & Prejudice          10       1 is
## # ... with 122,194 more rows
```

Ora, possiamo usare `inner_join()` per calcolare il sentimento in modi diversi.

Ricordate dall'alto che il lessico AFINN misura il sentimento con un punteggio numerico compreso tra -5 e 5, mentre gli altri due lessici categorizzano le parole in modo binario, sia positivo che negativo. Per trovare un punteggio sentimentale in blocchi di testo in tutto il romanzo, dovremo utilizzare un modello diverso per il lessico AFINN che per gli altri due.

Facciamo ancora una volta utilizzare divisione intera (%/%) per definire più grandi sezioni di testo che si estendono su più righe, e siamo in grado di utilizzare lo stesso modello con `count()`, `spread()` e `mutate()`di trovare il sentimento netta in ciascuna di queste sezioni di testo.

```
afinn <- pride_prejudice %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = linenumber %/% 80) %>%
  summarise(sentiment = sum(score)) %>%
  mutate(method = "AFINN")

bing_and_nrc <- bind_rows(pride_prejudice %>%
                            inner_join(get_sentiments("bing")) %>%
                            mutate(method = "Bing et al."),
                          pride_prejudice %>%
                            inner_join(get_sentiments("nrc") %>%
                                         filter(sentiment %in% c("positive",

"negative"))) %>%
                            mutate(method = "NRC")) %>%
  count(method, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

Ora abbiamo una stima del sentimento netto (positivo - negativo) in ogni pezzo del testo originale per ogni lessico di sentimento. Leggiamoli insieme e visualizzali nella Figura 2.3.

```
bind_rows(afinn,
          bing_and_nrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```

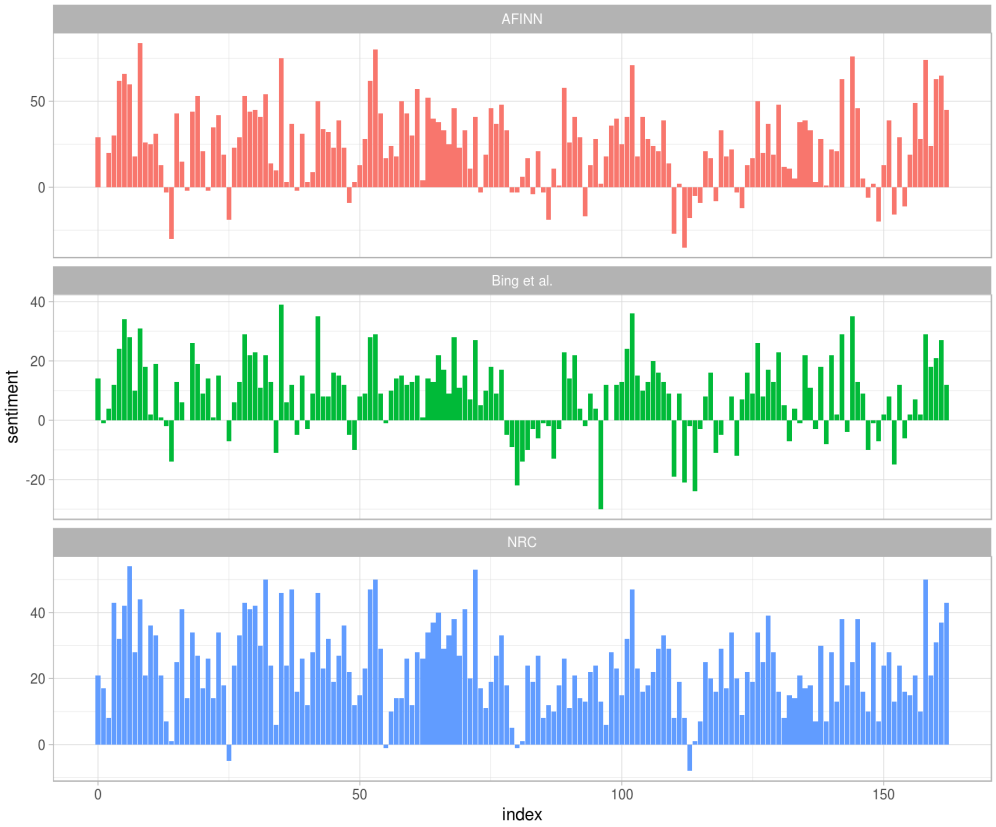


Figura 2.3: Confronto di tre lessici di sentimento usando Pride e Prejudice

I tre diversi lessici per calcolare il sentimento danno risultati diversi in senso assoluto ma hanno traiettorie relative simili attraverso il romanzo. Vediamo avvallamenti e picchi simili nei sentimenti intorno agli stessi punti del romanzo, ma i valori assoluti sono significativamente diversi. Il lessico AFINN fornisce i maggiori valori assoluti, con valori positivi elevati. Il lessico di Bing et al. ha valori assoluti più bassi e sembra etichettare blocchi più grandi di testo positivo o negativo contiguo. I risultati NRC sono spostati più in alto rispetto agli altri due, etichettando il testo in modo più positivo, ma rilevando cambiamenti relativi simili nel testo. Troviamo differenze simili tra i metodi quando guardiamo altri romanzi; il sentimento NRC è alto, il sentimento AFINN ha più varianza, il Bing et al.

Perché, ad esempio, il risultato per il lessico NRC è polarizzato così in alto rispetto al Bing et al. risultato? Diamo uno sguardo breve a quante parole positive e negative ci sono in questi lessici.

```
get_sentiments("nrc") %>%
     filter(sentiment %in% c("positive",
                             "negative")) %>%
  count(sentiment)
## # A tibble: 2 x 2
```

```
##   sentiment     n
##   <chr>     <int>
## 1 negative   3324
## 2 positive   2312
```

```
get_sentiments("bing") %>%
  count(sentiment)
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   4782
## 2 positive   2006
```

Entrambi i lessici hanno più parole negative che positive, ma il rapporto tra le parole negative e positive è più elevato nel lessico di Bing rispetto al lessico NRC. Ciò contribuirà all'effetto che vediamo nella trama sopra, così come qualsiasi differenza sistematica nella corrispondenza delle parole, ad esempio se le parole negative nel lessico NRC non corrispondono alle parole che Jane Austen utilizza molto bene. Qualunque sia la fonte di queste differenze, vediamo traiettorie relative simili attraverso l'arco narrativo, con cambiamenti simili nella pendenza, ma differenze marcate nel sentimento assoluto dal lessico al lessico. Questo è un contesto importante da tenere a mente quando si sceglie un lessico di sentimenti per l'analisi.

5.2.4 Parole positive e negative più comuni

Un vantaggio di avere il frame dei dati sia con il *sentiment* che con *word* è che possiamo analizzare i conteggi delle parole che contribuiscono a ciascun sentimento. Implementando `count()` qui con argomenti di entrambi `word` e `sentiment`, scopriamo quanto ogni parola ha contribuito a ciascun sentimento.

```
bing_word_counts <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

bing_word_counts
```

```
## # A tibble: 2,585 x 3
##    word      sentiment     n
##    <chr>     <chr>     <int>
##  1 miss      negative   1855
##  2 well      positive   1523
##  3 good      positive   1380
##  4 great     positive    981
##  5 like      positive    725
##  6 better    positive    639
##  7 enough    positive    613
##  8 happy     positive    534
##  9 love      positive    495
## 10 pleasure  positive    462
## # ... with 2,575 more rows
```

```
## # A tibble: 2,585 x 3
```

```
##    word      sentiment     n
##    <chr>     <chr>     <int>
##  1 miss      negative   1855
##  2 well      positive   1523
##  3 good      positive   1380
##  4 great     positive    981
##  5 like      positive    725
##  6 better    positive    639
##  7 enough    positive    613
##  8 happy     positive    534
##  9 love      positive    495
## 10 pleasure  positive    462
## # ... with 2,575 more rows
```

Questo può essere mostrato visivamente, e, se vogliamo, possiamo canalizzare direttamente in ggplot2, per il modo in cui usiamo costantemente gli strumenti creati per gestire i frame di dati in ordine.

```
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()
```

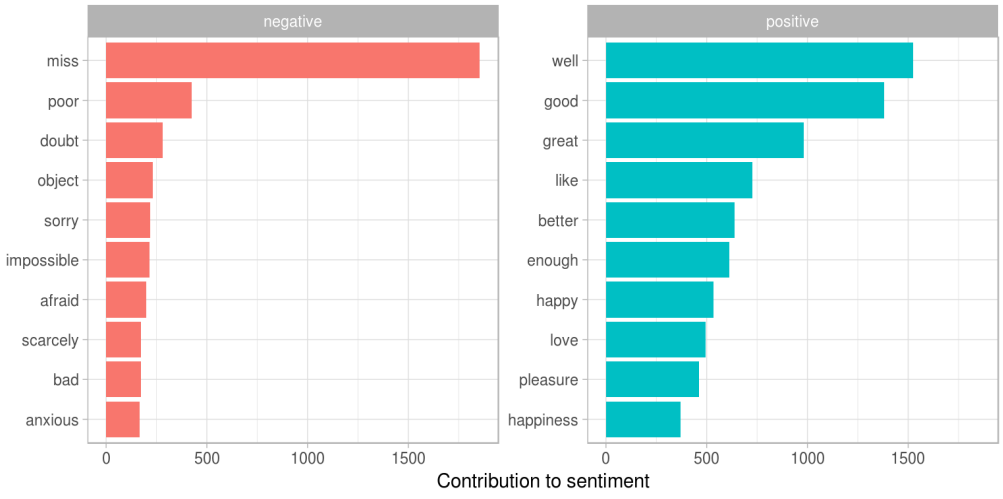


Figura 2.4: Parole che contribuiscono al sentimento positivo e negativo nei romanzi di Jane Austen

La figura 2.4 ci consente di individuare un'anomalia nell'analisi dei sentimenti; la parola "miss" è codificata come negativa ma è usata come titolo per giovani donne non sposate nelle opere di Jane Austen. Se fosse appropriato per i nostri scopi, potremmo facilmente aggiungere "miss" a un elenco di parole-stop personalizzato usando `bind_rows()`. Potremmo implementarlo con una strategia come questa.

```
custom_stop_words <- bind_rows(data_frame(word = c("miss"),
                                          lexicon = c("custom")),
```

```
                                stop_words)

custom_stop_words
```

```
## # A tibble: 1,150 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 miss        custom
##  2 a           SMART
##  3 a's         SMART
##  4 able        SMART
##  5 about       SMART
##  6 above       SMART
##  7 according   SMART
##  8 accordingly SMART
##  9 across      SMART
## 10 actually    SMART
## # ... with 1,140 more rows
```

5.2.5 Wordclouds

Abbiamo visto che questo metodo di text mining ordinato funziona bene con ggplot2, ma avere i nostri dati in un formato ordinato è utile anche per altri grafici.

Ad esempio, considera il pacchetto wordcloud, che utilizza la grafica di base R. Diamo un'occhiata alle parole più comuni nelle opere di Jane Austen nel suo complesso, ma questa volta come un wordcloud nella Figura 2.5 .

```
library(wordcloud)

tidy_books %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

Figura 2.5: Le parole più comuni nei romanzi di Jane Austen

In altre funzioni, ad esempio `comparison.cloud()`, potrebbe essere necessario trasformare il frame di dati in una matrice con reshape2 `acast()`. Facciamo l'analisi del sentimento per taggare le parole positive e negative usando un inner join, quindi trova le parole positive e negative più comuni. Fino al passaggio in cui dobbiamo inviare i dati `comparison.cloud()`, tutto questo può essere fatto con join, piping e dplyr perché i nostri dati sono in un formato ordinato.

```
library(reshape2)

tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("gray20", "gray80"),
                   max.words = 100)
```

Figura 2.6: Parole positive e negative più comuni nei romanzi di Jane Austen

La dimensione del testo di una parola nella figura 2.6 è proporzionale alla sua frequenza all'interno del suo sentimento. Possiamo usare questa visualizzazione per vedere le parole positive e negative più importanti, ma le dimensioni delle parole non sono confrontabili tra i sentimenti.

5.2.6 Guardare le unità oltre le sole parole

Un sacco di lavoro utile può essere svolto mediante tokenizzazione a livello di parola, ma a volte è utile o necessario esaminare diverse unità di testo. Ad esempio, alcuni algoritmi di analisi del sentimento guardano al di là solo degli unigrams (cioè parole singole) per cercare di comprendere il sentimento di una frase nel suo complesso. Questi algoritmi cercano di capirlo

Non sto vivendo una buona giornata.

è una frase triste, non felice, a causa della negazione. I pacchetti R includevano coreNLP. cleanNLP e sentimentr sono esempi di tali algoritmi di analisi del sentimento. Per questi, potremmo voler tokenizzare il testo in frasi, e ha senso usare un nuovo nome per la colonna di output in questo caso.

```
PandP_sentences <- data_frame(text = prideprejudice) %>%
  unnest_tokens(sentence, text, token = "sentences")
```

Diamo un'occhiata a uno solo.

```
PandP_sentences$sentence[2]
```

```
## [1] "however little known the feelings or views of such a man may be on
his first entering a neighbourhood, this truth is so well fixed in the minds
```

```
of the surrounding families, that he is considered the rightful property of
some one or other of their daughters."
```

La tokenizzazione della frase sembra avere un po' di problemi con il testo codificato UTF-8, specialmente con le sezioni di dialogo; fa molto meglio con la punteggiatura in ASCII. Una possibilità, se questo è importante, è provare a utilizzare `iconv()`, con qualcosa di simile `iconv(text, to = 'latin1')` in un'affermazione mutante, prima di essere inanimato.

Un'altra opzione `unnest_tokens()` è quella di dividere in token usando un modello regex. Potremmo usare questo, ad esempio, per dividere il testo dei romanzi di Jane Austen in un frame di dati per capitolo.

```
austen_chapters <- austen_books() %>%
  group_by(book) %>%
  unnest_tokens(chapter, text, token = "regex",
                pattern = "Chapter|CHAPTER [\\dIVXLC]") %>%
  ungroup()

austen_chapters %>%
  group_by(book) %>%
  summarise(chapters = n())
```

```
## # A tibble: 6 x 2
##   book                chapters
##   <fct>                  <int>
## 1 Sense & Sensibility       51
## 2 Pride & Prejudice         62
## 3 Mansfield Park            49
## 4 Emma                      56
## 5 Northanger Abbey          32
## 6 Persuasion                25
```

Abbiamo recuperato il numero corretto di capitoli in ciascun romanzo (più una riga "extra" per ogni titolo del romanzo). Nel DF `austen_chapters`, ogni riga corrisponde a un capitolo.

All'inizio di questo capitolo, abbiamo usato una regex simile per trovare dove tutti i capitoli erano nei romanzi di Austen per un tidy DF organizzato da una parola per riga. Possiamo usare l'analisi del testo ordinata per porre domande su quali sono i capitoli più negativi in ciascuno dei romanzi di Jane Austen? Per prima cosa, prendiamo l'elenco di parole negative dal lessico di Bing. In secondo luogo, facciamo un DF di quante parole ci sono in ogni capitolo in modo che possiamo normalizzare per la lunghezza dei capitoli. Quindi, troviamo il numero di parole negative in ciascun capitolo e dividiamo per le parole totali in ciascun capitolo. Per ogni libro, quale capitolo ha la percentuale più alta di parole negative?

```
bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")

wordcounts <- tidy_books %>%
  group_by(book, chapter) %>%
  summarize(words = n())

tidy_books %>%
  semi_join(bingnegative) %>%
  group_by(book, chapter) %>%
```

```
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c("book", "chapter")) %>%
  mutate(ratio = negativewords/words) %>%
  filter(chapter != 0) %>%
  top_n(1) %>%
  ungroup()
```

```
## # A tibble: 6 x 5
##   book                chapter negativewords words  ratio
##   <fct>                 <int>         <int> <int>  <dbl>
## 1 Sense & Sensibility      43           161  3405 0.0473
## 2 Pride & Prejudice        34           111  2104 0.0528
## 3 Mansfield Park           46           173  3685 0.0469
## 4 Emma                     15           151  3340 0.0452
## 5 Northanger Abbey         21           149  2982 0.0500
## 6 Persuasion                4            62  1807 0.0343
```

Questi sono i capitoli con le parole più tristi in ogni libro, normalizzati per il numero di parole nel capitolo. Cosa sta succedendo in questi capitoli? Nel capitolo 43 di Sense and Sensibility Marianne è gravemente ammalata, prossima alla morte, e nel capitolo 34 di Pride and Prejudice Mr. Darcy propone per la prima volta (così male!). Il capitolo 46 di Mansfield Park è quasi alla fine, quando tutti conoscono lo scandaloso adulterio di Henry, il capitolo 15 di Emma è quando l'orribile Mr. Elton propone, e nel capitolo 21 di Northanger Abbey Catherine è immersa nella sua finta fantasia gotica di omicidio, ecc. Capitolo 4 di Persuasione è quando il lettore riceve il pieno flashback di Anne che rifiuta il Capitano Wentworth e di quanto sia triste e che terribile errore abbia realizzato.

5.2.7 Riepilogo

L'analisi del sentimento fornisce un modo per comprendere gli atteggiamenti e le opinioni espresse nei testi. In questo capitolo, abbiamo esplorato come affrontare l'analisi del sentiment usando i principi dei dati in ordine; quando i dati di testo si trovano in un DF tidy, la sentiment analysis può essere implementata come inner join. Possiamo usare l'analisi del sentimento per capire come un arco narrativo cambia durante il suo corso o quali parole con contenuto emotivo e di opinione sono importanti per un particolare testo.

3 Analizzare la frequenza di parole e documenti: tf-idf

TECNICA TF-IDF. Altra cosa che possiamo fare è calcolare la statistica tf-idf. La statistica tf-idf ha lo scopo di misurare quanto sia importante una parola per un documento in una raccolta (o un corpus) di documenti, per esempio, in un romanzo, o in un auna raccolta di romanzi o in un sito web in una raccolta di siti web.

- **Tf** (*term frequency*) indica la frequenza di una parola in un documento. Ma se una parola è frequente anche in altri documenti allora non caratterizza quel documento; devo pesare quanto una parola è specifica per quel documento
- Moltiplico la tf per l'**Idf** (*inverse document frequency*), è il log di n/nt

Se t è un termine, **nt** è il numero di documenti che contengono quel termine e n sarà il numero di documenti. Quindi se tutti i documenti contengono quel termine n sarà = nt quindi il rapporto è 1 e il log sarà 0, annullando la frequenza.

$$idf(\text{term}) = \ln(n_{\text{documents}}/n_{\text{documents containing term}})$$

Quindi il termine deve essere presente in quel documento e deve essere specifico di pochi documenti.

Una questione centrale nell'estrazione del testo e nell'elaborazione del linguaggio naturale è come quantificare di cosa tratta un documento. Possiamo farlo guardando le parole che compongono il documento? Una misura di quanto può essere importante una parola è la frequenza del termine (**tf**) cioè quanto frequentemente si verifica una parola in un documento. Ci sono parole in un documento, tuttavia, che si presentano molte volte ma potrebbero non essere importanti; in italiano, queste sono probabilmente parole come "il", "è", "di", e così via. Potremmo adottare l'approccio di aggiungere parole come queste a un elenco di parole chiave e rimuoverle prima di fare analisi, ma è possibile che alcune di queste parole potrebbero essere più importanti in alcuni documenti rispetto ad altri. Un elenco di parole chiave non è un approccio molto sofisticato per regolare la frequenza dei termini per le parole di uso comune.

Un altro approccio è quello di esaminare la frequenza inversa del documento (**idf**) di un termine, che diminuisce il peso per le parole di uso comune e aumenta il peso per le parole che non vengono utilizzate molto in una raccolta di documenti. Questo può essere combinato con la frequenza dei termini per calcolare il termine *tf-idf* di un termine (le due quantità moltiplicate insieme), la frequenza di un termine aggiustata per quanto raramente viene utilizzata.

La statistica **tf-idf** ha lo scopo di misurare quanto sia importante una parola per un documento in una raccolta (o un corpus) di documenti, per esempio, in un romanzo di una raccolta di romanzi o in un sito web in una raccolta di siti web.

È una regola empirica o euristica; mentre si è dimostrato utile nel text mining, nei motori di ricerca, ecc., i suoi fondamenti teorici sono considerati non molto stabili dagli esperti di teoria dell'informazione. La frequenza inversa del documento per ogni dato termine è definita come

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

Possiamo utilizzare i principi dei dati in ordine per approcciare l'analisi tf-idf e utilizzare strumenti coerenti ed efficaci per quantificare l'importanza dei vari termini in un documento che fa parte di una raccolta.

5.3.1 Frequenza dei termini nei romanzi di Jane Austen

Cominciamo guardando i romanzi pubblicati di Jane Austen ed esaminiamo la frequenza del primo termine, quindi tf-idf. Possiamo iniziare semplicemente usando i verbi *dplyr* come *group_by()* e *join()*. Quali sono le parole più usate nei romanzi di Jane Austen? (Calcoliamo anche le parole totali in ciascun romanzo qui, per un uso successivo.)

```
library(dplyr)
library(janeaustenr)
library(tidytext)

book_words <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE) %>%
  ungroup()
```

```
total_words <- book_words %>%
  group_by(book) %>%
  summarize(total = sum(n))

book_words <- left_join(book_words, total_words)

book_words
## # A tibble: 40,379 x 4
##    book              word      n  total
##    <fct>             <chr> <int>  <int>
##  1 Mansfield Park    the    6206 160460
##  2 Mansfield Park    to     5475 160460
##  3 Mansfield Park    and    5438 160460
##  4 Emma              to     5239 160996
##  5 Emma              the    5201 160996
##  6 Emma              and    4896 160996
##  7 Mansfield Park    of     4778 160460
##  8 Pride & Prejudice the    4331 122204
##  9 Emma              of     4291 160996
## 10 Pride & Prejudice to     4162 122204
## # ... with 40,369 more rows
```

C'è una riga in questo `book_wordsframe` di dati per ogni combinazione di word-book; `n` è il numero di volte in cui quella parola è usata in quel libro e `total` sono le parole totali in quel libro. I soliti sospetti sono qui con il più alto n, "il", "e", "a", e così via. Nella Figura 3.1, diamo un'occhiata alla distribuzione di ogni romanzo `n/total`: il numero di volte in cui una parola appare in un romanzo divisa per il numero totale di termini (parole) in quel romanzo. Questo è esattamente il termine frequenza.

```
library(ggplot2)

ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
```

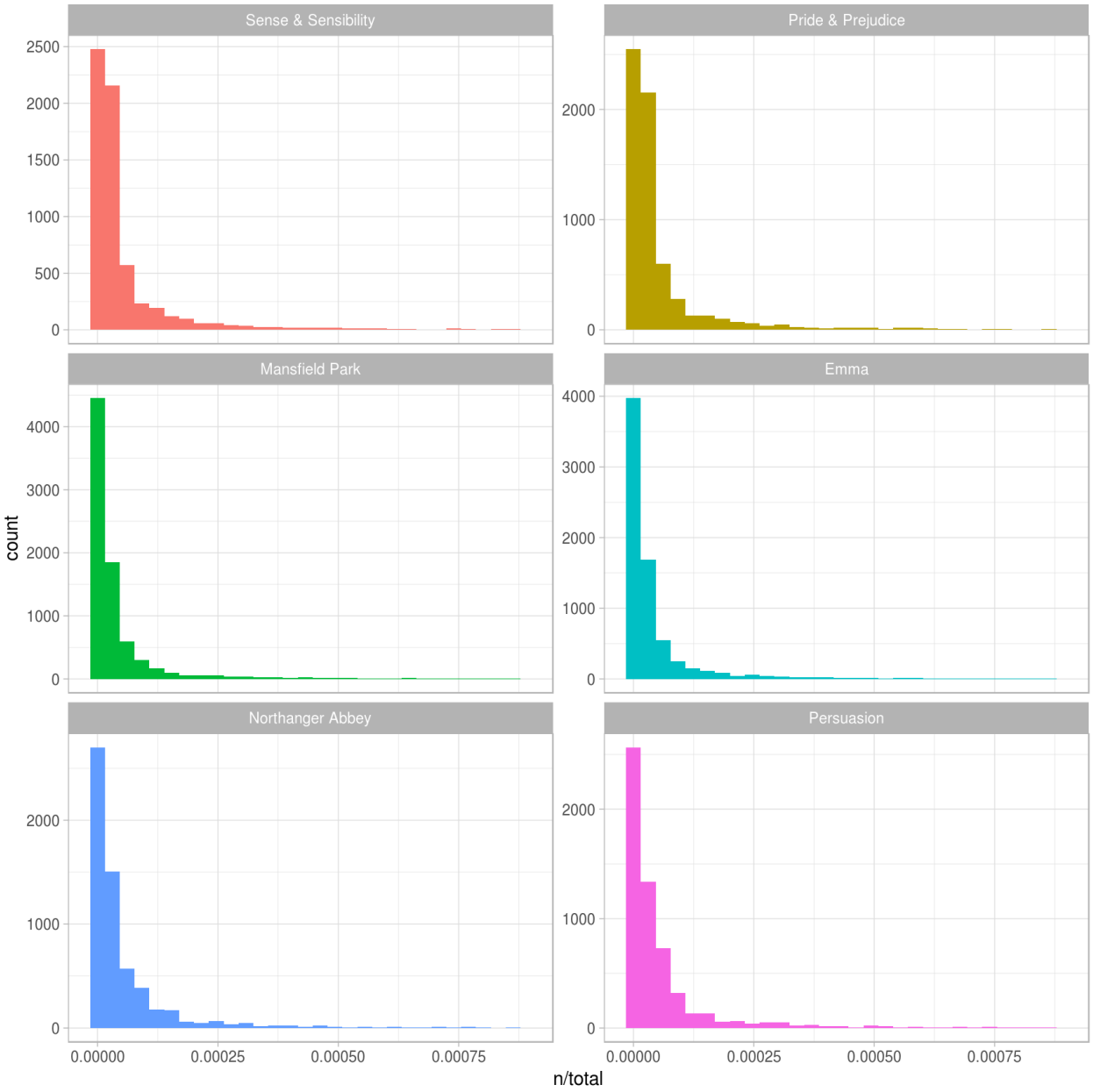


Figura 3.1: Distribuzione della frequenza dei termini nei romanzi di Jane Austen (in pratica sono frequenze relative)

Ci sono code molto lunghe a destra per questi romanzi (quelle parole estremamente comuni!) Che non abbiamo mostrato in questi grafici. Questi grafici mostrano distribuzioni simili per tutti i romanzi, con molte parole che si verificano raramente e poche parole che si verificano più frequentemente.

5.3.2 Legge di Zipf

Legge di Zipf, legge empirica. Se prendiamo un qualsiasi testo e calcoliamo la frequenza delle parole all'interno del testo si osserva che la frequenza di quella parola è inversamente proporzionale al suo rango. Ordiniamo le parole per frequenza, ci sarà un rank, una parola ed una frequenza come variabili. Se la frequenza della prima parola è p1 allora la frequenza della seconda parola sarà la metà, la frequenza della terza parola sarà un terzo rispetto alla prima ecc.

Le distribuzioni come quelle mostrate nella Figura 3.1 sono tipiche nella lingua. In effetti, quei tipi di distribuzioni a coda lunga sono così comuni in ogni dato corpus di linguaggio naturale (come un libro, o molto testo da un sito Web, o parole parlate) che la relazione tra la frequenza con cui viene usata una parola e il suo rango è stato oggetto di studio; una versione classica di questa relazione è chiamata legge di Zipf, da George Zipf, un linguista americano del XX secolo.

La legge di Zipf afferma che la frequenza di visualizzazione di una parola è inversamente proporzionale alla sua posizione.

Dato che abbiamo il DF usato per tracciare la frequenza dei termini, possiamo esaminare la legge di Zipf per i romanzi di Jane Austen con poche righe di codice con dplyr.

```
freq_by_rank <- book_words %>%
  group_by(book) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)

freq_by_rank
## # A tibble: 40,379 x 6
## # Groups:   book [6]
##    book              word      n  total  rank `term frequency`
##    <fct>             <chr> <int>  <int> <int>            <dbl>
##  1 Mansfield Park    the    6206 160460     1           0.0387
##  2 Mansfield Park    to     5475 160460     2           0.0341
##  3 Mansfield Park    and    5438 160460     3           0.0339
##  4 Emma              to     5239 160996     1           0.0325
##  5 Emma              the    5201 160996     2           0.0323
##  6 Emma              and    4896 160996     3           0.0304
##  7 Mansfield Park    of     4778 160460     4           0.0298
##  8 Pride & Prejudice the    4331 122204     1           0.0354
##  9 Emma              of     4291 160996     4           0.0267
## 10 Pride & Prejudice to     4162 122204     2           0.0341
## # ... with 40,369 more rows
```

La colonna `rank` qui ci dice il rango di ogni parola all'interno della tabella delle frequenze; la tabella era già stata ordinata in `n` modi da poterla usare per trovare il rango `row_number()`. Quindi, possiamo calcolare la frequenza frl termine nello stesso modo in cui lo abbiamo fatto prima. La legge di Zipf viene spesso visualizzata tracciando un rango sull'asse x e la frequenza dei termini sull'asse y, su scale logaritmiche. Disegnandola così, una relazione inversamente proporzionale avrà una pendenza negativa costante.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```

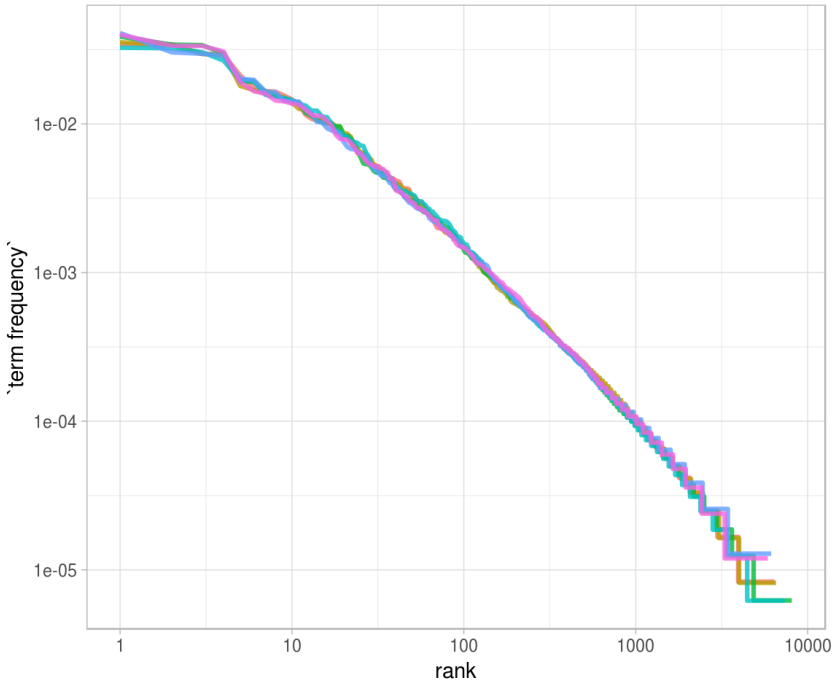


Figura 3.2: La legge di Zipf per i romanzi di Jane Austen

Si noti che la Figura 3.2 si trova in coordinate log-log. Vediamo che tutti e sei i romanzi di Jane Austen sono simili tra loro e che la relazione tra rango e frequenza ha una pendenza negativa. Non è abbastanza costante, però; forse potremmo vederlo come una legge di potenza spezzata con, diciamo, tre sezioni. Vediamo quale è l'esponente della legge di potenza per la sezione centrale dell'intervallo di gradi.

```
rank_subset <- freq_by_rank %>%
  filter(rank < 500,
         rank > 10)

lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -0.6226      -1.1125
```

Le versioni classiche della legge di Zipf hanno

$$\text{frequency} \propto \frac{1}{\text{rank}}$$

e in effetti abbiamo ottenuto una pendenza vicina a -1 qui. Analizziamo questa legge di potenza adattata con i dati nella Figura 3.3 per vedere come appare.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_abline(intercept = -0.62, slope = -1.1, color = "gray50", linetype =
2) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
```

```
  scale_x_log10() +
  scale_y_log10()
```

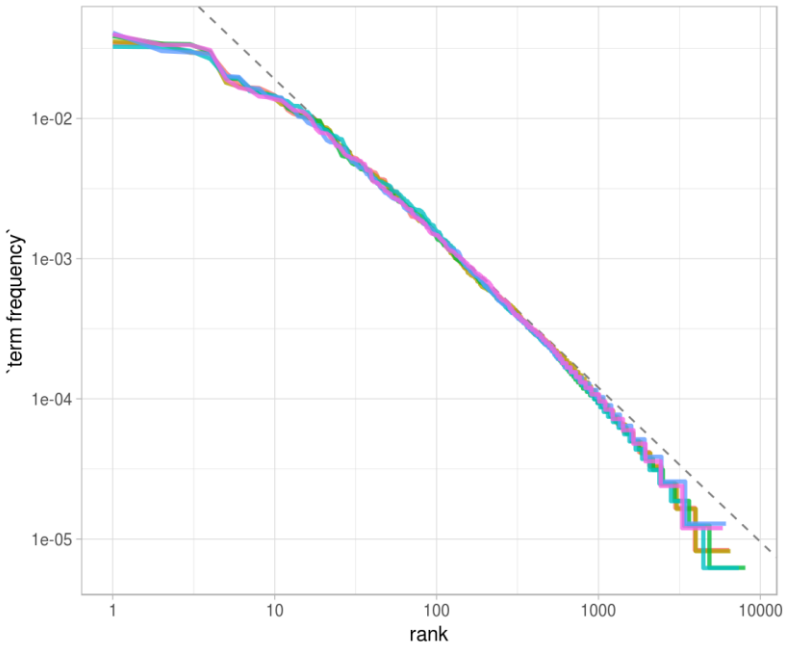


Figura 3.3: Adattare un esponente per la legge di Zipf con i romanzi di Jane Austen

Abbiamo trovato un risultato vicino alla versione classica della legge di Zipf per il corpus dei romanzi di Jane Austen. Le deviazioni che vediamo qui in alto grado non sono rare per molti tipi di linguaggio; un corpus di linguaggio spesso contiene meno parole rare di quanto previsto da una singola legge di potenza. Le deviazioni a basso rango sono più insolite. Jane Austen usa una percentuale inferiore delle parole più comuni di molte raccolte di lingue. Questo tipo di analisi potrebbe essere esteso per confrontare gli autori o per confrontare qualsiasi altra raccolta di testo; può essere implementato semplicemente usando i principi dei tidy data.

5.3.3 La funzione bind_tf_idf()

`C'è una funzione che lavora su un dataframe che contiene delle variabili (word, book, n che tiene conto di quante volte quella parola occorre in quel documento): bind_tf_idf e calcola il tf-idf.`

L'idea di tf-idf è di trovare le parole importanti per il contenuto di ogni documento diminuendo il peso per le parole di uso comune e aumentando il peso per le parole che non vengono utilizzate molto in una raccolta o in un corpus di documenti, in questo caso, il gruppo dei romanzi di Jane Austen nel suo insieme. Calcolando tf-idf tenta di trovare le parole che sono importanti (cioè comuni) in un testo, ma non troppo comuni. Facciamolo ora.

La funzione `bind_tf_idf` nel pacchetto tidytext richiede un dataset tidy come input con una riga per token (termine), per ciascun documento. Una colonna (qui `word`) contiene i termini / token, una colonna contiene i documenti (in questo caso `book`) e l'ultima colonna necessaria contiene i conteggi, cioè quante volte ogni documento contiene ciascun termine (`n` in questo esempio). Abbiamo calcolato `total` per ogni libro per le nostre esplorazioni nelle sezioni precedenti, ma non è necessario per la funzione `bind_tf_idf`; la tabella deve contenere solo tutte le parole in ogni documento.

```
book_words <- book_words %>%
```

```
  bind_tf_idf(word, book, n)
book_words
## # A tibble: 40,379 x 7
##    book              word      n  total     tf   idf tf_idf
##    <fct>             <chr> <int>  <int>  <dbl> <dbl>  <dbl>
##  1 Mansfield Park    the    6206 160460 0.0387     0      0
##  2 Mansfield Park    to     5475 160460 0.0341     0      0
##  3 Mansfield Park    and    5438 160460 0.0339     0      0
##  4 Emma              to     5239 160996 0.0325     0      0
##  5 Emma              the    5201 160996 0.0323     0      0
##  6 Emma              and    4896 160996 0.0304     0      0
##  7 Mansfield Park    of     4778 160460 0.0298     0      0
##  8 Pride & Prejudice the    4331 122204 0.0354     0      0
##  9 Emma              of     4291 160996 0.0267     0      0
## 10 Pride & Prejudice to     4162 122204 0.0341     0      0
## # ... with 40,369 more rows
```

Si noti che idf e quindi tf-idf sono zero per queste parole estremamente comuni. Queste sono tutte parole che appaiono in tutti e sei i romanzi di Jane Austen, quindi il termine idf (che sarà quindi il log naturale di 1) è zero. La frequenza inversa del documento (e quindi tf-idf) è molto bassa (vicino allo zero) per le parole che si verificano in molti dei documenti di una raccolta; questo è il modo in cui questo approccio riduce il peso per le parole comuni. La frequenza inversa del documento sarà più alta per le parole che si verificano in meno dei documenti nella raccolta.

Diamo un'occhiata ai termini con alta tf-idf nelle opere di Jane Austen

```
book_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 40,379 x 6
##    book                word          n      tf   idf  tf_idf
##    <fct>               <chr>     <int>   <dbl> <dbl>   <dbl>
##  1 Sense & Sensibility elinor      623 0.00519  1.79 0.00931
##  2 Sense & Sensibility marianne    492 0.00410  1.79 0.00735
##  3 Mansfield Park      crawford    493 0.00307  1.79 0.00551
##  4 Pride & Prejudice   darcy       373 0.00305  1.79 0.00547
##  5 Persuasion          elliot      254 0.00304  1.79 0.00544
##  6 Emma                emma        786 0.00488  1.10 0.00536
##  7 Northanger Abbey    tilney      196 0.00252  1.79 0.00452
##  8 Emma                weston      389 0.00242  1.79 0.00433
##  9 Pride & Prejudice   bennet      294 0.00241  1.79 0.00431
## 10 Persuasion          wentworth   191 0.00228  1.79 0.00409
## # ... with 40,369 more rows
```

Qui vediamo tutti i nomi propri, nomi che sono importanti in questi romanzi. Nessuno di essi compare in tutti i romanzi e sono parole importanti e caratteristiche per ogni testo all'interno del corpus dei romanzi di Jane Austen.

Alcuni dei valori di idf sono gli stessi per termini diversi perché ci sono 6 documenti in questo corpus e stiamo vedendo il valore numerico per $\ln(6/1)$, ln(6/1), $\ln(6/2)$, ln(6/2), eccetera.

Diamo un'occhiata a una visualizzazione per queste parole alte tf-idf nella Figura 3.4.

```
book_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
```

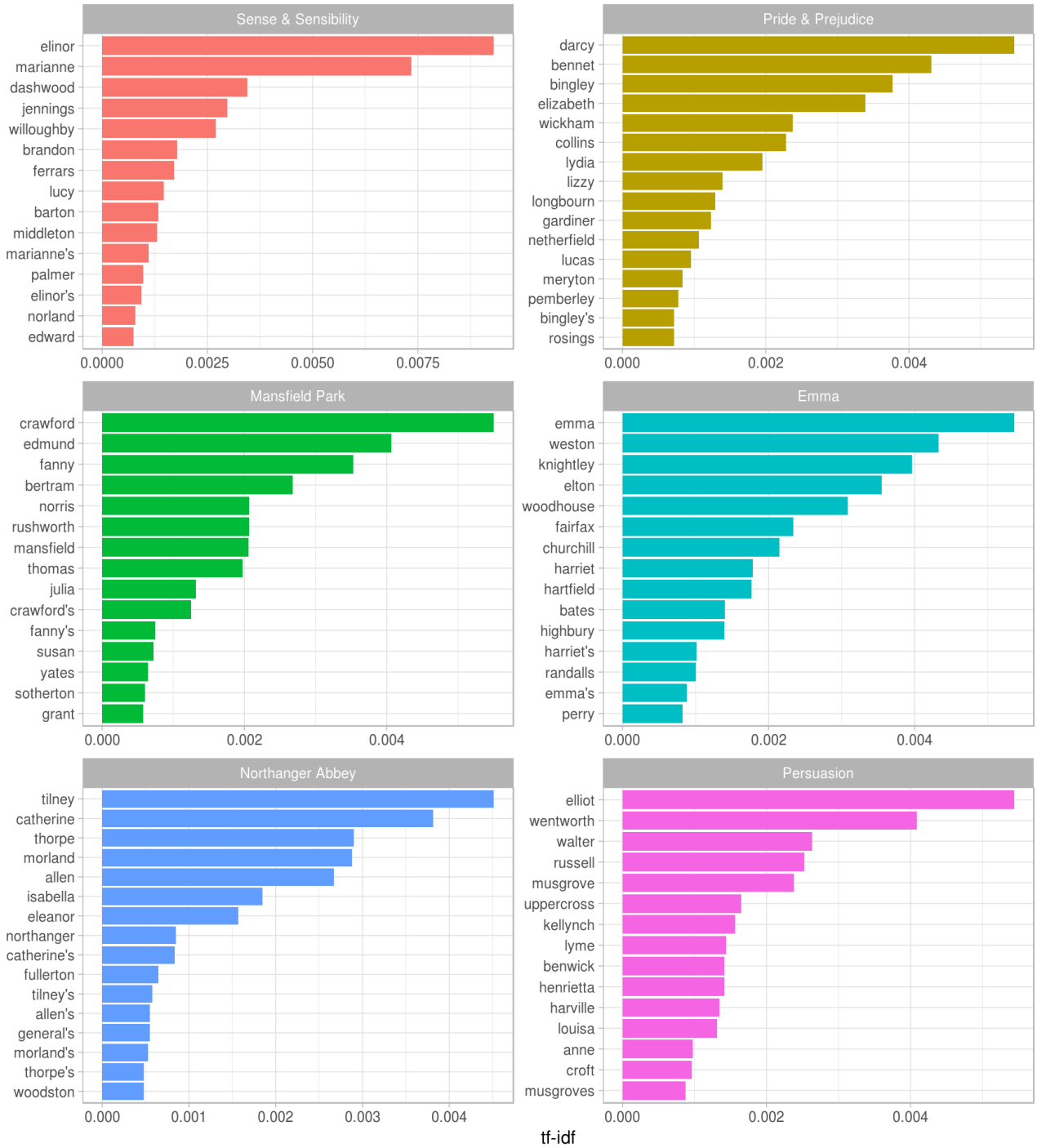


Figura 3.4: Le parole più alte di tf-idf in ciascuno dei romanzi di Jane Austen

Ancora tutti i nomi propri nella Figura 3.4 ! Queste parole sono, come misurato da tf-idf, il più importante per ogni romanzo e la maggior parte dei lettori sarebbe probabilmente d'accordo. Ciò che la misurazione di tf-idf ha fatto qui ci mostra che Jane Austen ha usato un linguaggio simile nei suoi sei romanzi, e ciò che distingue un romanzo dal resto all'interno della collezione delle sue opere sono i nomi propri, i nomi delle persone e dei luoghi. Questo è il punto di tf-idf; identifica le parole che sono importanti per un documento all'interno di una raccolta di documenti.

5.3.4 Un corpus di testi di fisica

Omesso

5.3.5 Riepilogo

L'uso della frequenza dei termini e della frequenza inversa del documento ci consente di trovare parole che sono caratteristiche di un documento all'interno di una raccolta di documenti, sia che si tratti di un romanzo o di un testo fisico o di una pagina web. Esplorare la frequenza dei termini da sola può darci un'idea di come il linguaggio viene utilizzato in una raccolta di linguaggio naturale, e verbi dplyr come `count()` e `rank()` ci danno strumenti per ragionare sulla frequenza dei termini. Il pacchetto **tidytext** utilizza un'implementazione di tf-idf coerente con i principi dei tidy datache ci consente di vedere come le parole diverse sono importanti nei documenti all'interno di una raccolta o di un corpus di documenti.

4 Relazioni tra parole: n-grammi e correlazioni

N-GRAMS. Fino ad ora abbiamo lavorato con singole parole, ma noi ora vogliamo stabilire delle relazioni tra i termini, un grafo di similarità (come la rete terroristica degli attentati a Madrid). Relazioni tra parole. Un modo si chiama n-grammi e l'altro si chiama correlazione.

Un **n-gramma** è una sequenza consecutiva di n parole. Se capita spesso che una parola x sia seguita da una parola y allora vuol dire che c'è una relazione tra quelle parole, non è solo un caso (stessa cosa dei delfini). Vogliamo costruire un grafo diretto pesato, di relazioni tra parole, dove l'arco diretto vuol dire che due parole sono consecutive nel testo e il peso mi dice quante volte ho trovato queste due parole consecutive nel testo.

Altro modo per vedere la relazione tra due parole è guardare se queste sono molte volte nella stessa frase oppure nello stesso paragrafo (due parole possono essere in relazione anche se non sono consecutive). Il peso dell'arco ci dice il numero di volte in cui le due parole sono state viste assieme. Stessa cosa delle reti. Grafo bipartito (indiretto) in cui da una parte abbiamo le parole e dall'altra parte abbiamo i paragrafi, tracciamo un arco se una parola è contenuta in un paragrafo; stabilirò una relazione tra due parole se sono contenute almeno in uno stesso paragrafo (posso pesare questi archi con dei pesi che sono il numero di volte in cui queste parole occorrono assieme). Coefficiente di correlazione che misura una relazione binaria.

Finora abbiamo considerato le parole come singole unità e abbiamo considerato le loro relazioni con i sentimenti o con i documenti. Tuttavia, molte analisi di testo interessanti si basano sulle relazioni tra le parole, sia che si esaminino quali parole tendono a seguire immediatamente le altre, o che tendano a co-verificarsi all'interno degli stessi documenti.

In questo capitolo, esploreremo alcuni dei metodi offerti da tidytext per calcolare e visualizzare le relazioni tra le parole nel set di dati del testo. Questo include l'argomento `token = "ngrams"`, che **tokenizza** da coppie di parole adiacenti piuttosto che da singole. Introdurremo anche due nuovi pacchetti: **`ggraph`** , che estende ggplot2 per costruire i grafici di rete, e **`widyr`** , che calcola correlazioni e distanze fra coppie all'interno di un tidy dataframe. Insieme espandono la nostra cassetta degli attrezzi per esplorare il testo all'interno del quadro dei tidy data.

5.4.1 Tokenizzare gli n-gram

Abbiamo usato la funzione `unnest_tokens` per tokenizzare in parole o, a volte, in frasi, e questo era utile per i tipi di sentiment analyss e di frequenza che abbiamo fatto finora. Ma possiamo anche usare la funzione per tokenizzare in sequenze consecutive di parole, chiamate **n-grammi**. Vedendo quanto spesso la parola X è seguita dalla parola Y, possiamo quindi costruire un modello delle relazioni tra di loro.

Lo facciamo aggiungendo l'opzione `token = "ngrams"` a `unnest_tokens()`, e impostando il numero di parole `n` che vogliamo catturare in ogni n-grammo. Quando impostiamo `n` a 2, stiamo esaminando coppie di due parole consecutive, spesso chiamate "**bigram**":

```
library(dplyr)
library(tidytext)
library(janeaustenr)

austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

austen_bigrams
## # A tibble: 725,049 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility sense and
##  2 Sense & Sensibility and sensibility
##  3 Sense & Sensibility sensibility by
##  4 Sense & Sensibility by jane
##  5 Sense & Sensibility jane austen
##  6 Sense & Sensibility austen 1811
##  7 Sense & Sensibility 1811 chapter
##  8 Sense & Sensibility chapter 1
##  9 Sense & Sensibility 1 the
## 10 Sense & Sensibility the family
## # ... with 725,039 more rows
```

Questa struttura dati è ancora una variante del formato di testo ordinato. È strutturato come un token-per-row (con metadati aggiuntivi, come `book`, ancora conservati), ma ogni token ora rappresenta un bigram.

Si noti che questi bigram si sovrappongono: "sense and" è un token, mentre "and sensibility" è un altro token.

5.4.1.1 Conteggio e filtraggio di n-grammi

I nostri soliti strumenti ordinati si applicano altrettanto bene all'analisi n-grammi. Possiamo esaminare i bigram più comuni usando dplyr's `count()`:

```
austen_bigrams %>%
  count(bigram, sort = TRUE)
## # A tibble: 211,236 x 2
##    bigram       n
##    <chr>    <int>
##  1 of the    3017
##  2 to be     2787
##  3 in the    2368
##  4 it was    1781
##  5 i am      1545
```

```
##  6 she had   1472
##  7 of her    1445
##  8 to the    1387
##  9 she was   1377
## 10 had been  1299
## # ... with 211,226 more rows
```

Come ci si potrebbe aspettare, molti dei bigram più comuni sono coppie di parole comuni (non interessanti), come `of thee` e `to be`: quelle che chiamiamo "**stop-words**". Qui è utile usare la funzione `separate()` di Tidy, che divide una colonna in più colonne in base ad un delimitatore. Questo ci permette di separarlo in due colonne, "word1" e "word2", a quel punto possiamo rimuovere casi in cui una è una stop-word.

```
library(tidyr)

bigrams_separated <- austen_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
## # A tibble: 33,421 x 3
##    word1   word2         n
##    <chr>   <chr>     <int>
##  1 sir     thomas      287
##  2 miss    crawford    215
##  3 captain wentworth   170
##  4 miss    woodhouse   162
##  5 frank   churchill   132
##  6 lady    russell     118
##  7 lady    bertram     114
##  8 sir     walter      113
##  9 miss    fairfax     109
## 10 colonel brandon     108
## # ... with 33,411 more rows
```

Possiamo vedere che i nomi (sia primo che ultimo o con un saluto) sono le coppie più comuni nei libri di Jane Austen.

In altre analisi, potremmo voler lavorare con le parole ricombinate. la funzione di di tidyr `unite()` è l'opposta di `separate()` e ci consente di ricombinare le colonne in una sola. Quindi, "separate / filter / count / unite" ci permettono di trovare i bigram più comuni che non contengono stop-words.

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

```
## # A tibble: 44,784 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility jane austen
##  2 Sense & Sensibility austen 1811
##  3 Sense & Sensibility 1811 chapter
##  4 Sense & Sensibility chapter 1
##  5 Sense & Sensibility norland park
##  6 Sense & Sensibility surrounding acquaintance
##  7 Sense & Sensibility late owner
##  8 Sense & Sensibility advanced age
##  9 Sense & Sensibility constant companion
## 10 Sense & Sensibility happened ten
## # ... with 44,774 more rows
```

In altre analisi potresti essere interessato ai **trigram** più comuni, che sono sequenze consecutive di 3 parole. Possiamo trovarlo impostando `n = 3`:

```
austen_books() %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
## # A tibble: 8,757 x 4
##    word1     word2     word3         n
##    <chr>     <chr>     <chr>     <int>
##  1 dear      miss      woodhouse    23
##  2 miss      de        bourgh       18
##  3 lady      catherine de           14
##  4 catherine de        bourgh       13
##  5 poor      miss      taylor       11
##  6 sir       walter    elliot       11
##  7 ten       thousand  pounds       11
##  8 dear      sir       thomas       10
##  9 twenty    thousand  pounds        8
## 10 replied   miss      crawford      7
## # ... with 8,747 more rows
```

5.4.1.2 Analizzare i bigram

Questo formato one-bigram per riga è utile per le analisi esplorative del testo. Come semplice esempio, potremmo essere interessati alle "strade" più comuni menzionate in ogni libro:

```
bigrams_filtered %>%
  filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
## # A tibble: 34 x 3
##    book                word1         n
##    <fct>               <chr>     <int>
##  1 Sense & Sensibility berkeley      16
##  2 Sense & Sensibility harley        16
##  3 Northanger Abbey    pulteney      14
##  4 Northanger Abbey    milsom        11
```

```
##  5 Mansfield Park      wimpole        10
##  6 Pride & Prejudice   gracechurch     9
##  7 Sense & Sensibility conduit         6
##  8 Sense & Sensibility bond            5
##  9 Persuasion          milsom          5
## 10 Persuasion          rivers          4
## # ... with 24 more rows
```

Un bigram può anche essere trattato come un termine in un documento nello stesso modo in cui abbiamo trattato le singole parole. Ad esempio, possiamo guardare il tf-idf (capitolo 3) dei bigram attraverso i romanzi di Austen. Questi valori di tf-idf possono essere visualizzati all'interno di ogni libro, proprio come abbiamo fatto per le parole (Figura 4.1).

```
bigram_tf_idf <- bigrams_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf
```

```
## # A tibble: 36,217 x 6
##    book                bigram               n     tf   idf tf_idf
##    <fct>               <chr>            <int>  <dbl> <dbl>  <dbl>
##  1 Persuasion          captain wentworth   170 0.0299  1.79 0.0535
##  2 Mansfield Park      sir thomas          287 0.0287  1.79 0.0515
##  3 Mansfield Park      miss crawford       215 0.0215  1.79 0.0386
##  4 Persuasion          lady russell        118 0.0207  1.79 0.0371
##  5 Persuasion          sir walter          113 0.0198  1.79 0.0356
##  6 Emma                miss woodhouse      162 0.0170  1.79 0.0305
##  7 Northanger Abbey    miss tilney          82 0.0159  1.79 0.0286
##  8 Sense & Sensibility colonel brandon     108 0.0150  1.79 0.0269
##  9 Emma                frank churchill     132 0.0139  1.79 0.0248
## 10 Pride & Prejudice   lady catherine      100 0.0138  1.79 0.0247
## # ... with 36,207 more rows
```

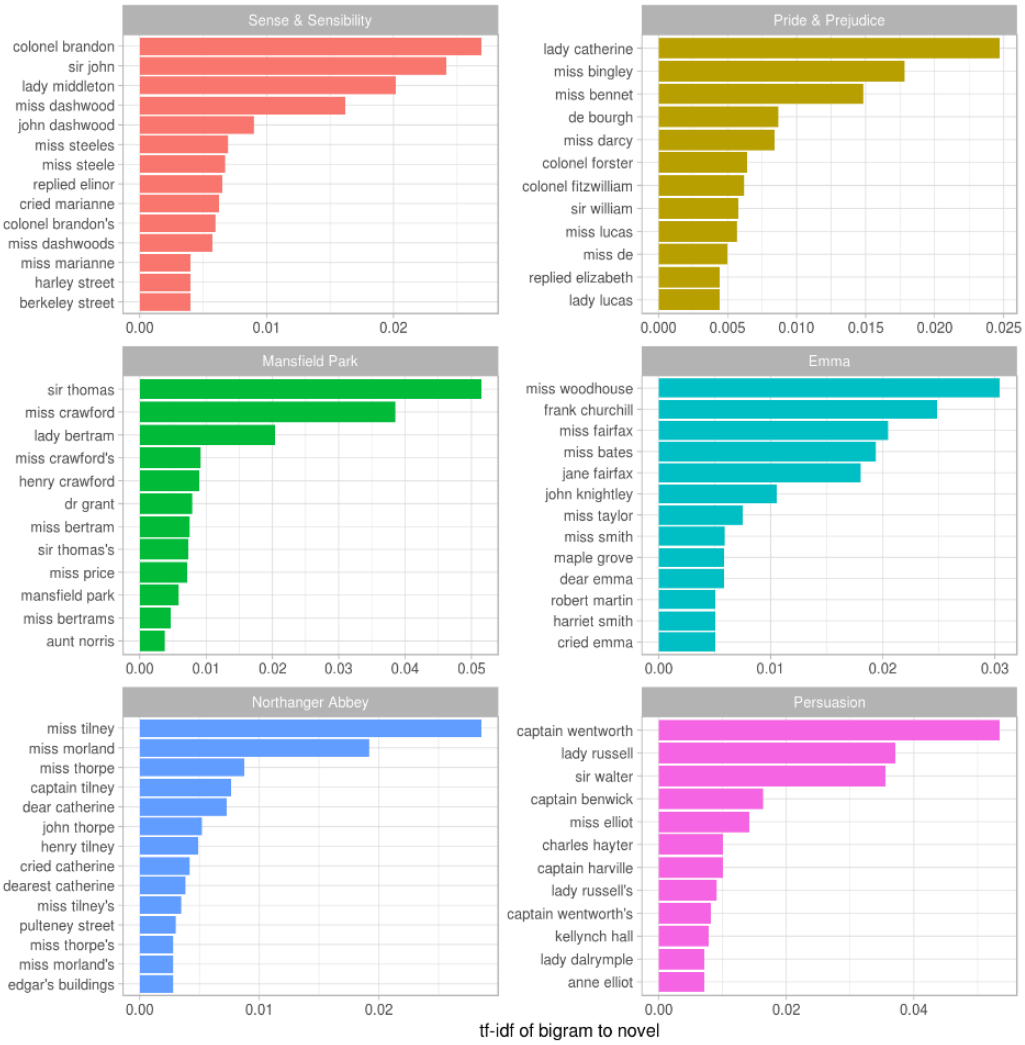


Figura 4.1: I 12 bigram con il più alto tf-idf di ogni romanzo di Jane Austen

Come abbiamo scoperto nel Capitolo 3 , le unità che distinguono ciascun libro di Austen sono quasi esclusivamente nomi. Notiamo anche alcuni abbinamenti di un verbo comune e un nome, come " replicò Elisabetta" in Orgoglio e Pregiudizio, o "emma emma" in Emma.
Ci sono vantaggi e svantaggi nell'esaminare il tf-idf dei bigram piuttosto che le singole parole. Coppie di parole consecutive potrebbero catturare una struttura che non è presente quando si contano solo parole singole e può fornire un contesto che rende più comprensibili i token (ad esempio, "pulteney street", nell'Abbazia di Northanger, è più informativo di "pulteney"). Tuttavia, i conteggi per-bigram sono anche più rari: una tipica coppia di due parole è più rara di una delle sue parole componenti. Pertanto, i bigram possono essere particolarmente utili quando si dispone di un set di dati di testo molto grande.

4.1.3 Usare i bigram per fornire un contesto nell'analisi dei sentimenti

Il nostro approccio di analisi del sentimento nel Capitolo 2 ha semplicemente contato l'aspetto di parole positive o negative, secondo un lessico di riferimento. Uno dei problemi con questo approccio è che il contesto di una parola può importare quasi quanto la sua presenza. Ad esempio, le parole "felice" e "mi piace" saranno contate come positive, anche in una frase del tipo "Non sono **felice** e non mi **piace**!"

Ora che i dati sono organizzati in bigram, è facile dire quanto spesso le parole siano precedute da una parola come "not":

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
```

```
## # A tibble: 1,246 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1 not   be      610
##  2 not   to      355
##  3 not   have    327
##  4 not   know    252
##  5 not   a       189
##  6 not   think   176
##  7 not   been    160
##  8 not   the     147
##  9 not   at      129
## 10 not   in      118
## # ... with 1,236 more rows
```

Eseguendo l'analisi del sentimento sui dati bigram, possiamo esaminare quanto spesso le parole associate al sentimento siano precedute da "non" o altre parole negative. Potremmo usare questo per ignorare o addirittura invertire il loro contributo al punteggio sentiment.

Usiamo il lessico AFINN per l'analisi dei sentimenti, che è possibile richiamare fornisce un punteggio numerico di sentimento per ogni parola, con numeri positivi o negativi che indicano la direzione del sentimento.

```
AFINN <- get_sentiments("afinn")

AFINN
## # A tibble: 2,476 x 2
##    word       score
##    <chr>      <int>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # ... with 2,466 more rows
```

Possiamo quindi esaminare le parole più frequenti che sono state precedute da "non" e sono state associate a un sentimento.

```
not_words <- bigrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()
```

```
not_words

## # A tibble: 245 x 3
##    word2   score     n
##    <chr>   <int> <int>
##  1 like        2    99
##  2 help        2    82
##  3 want        1    45
##  4 wish        1    39
##  5 allow       1    36
##  6 care        2    23
##  7 sorry      -1    21
##  8 leave      -1    18
##  9 pretend    -1    18
## 10 worth       2    17
## # ... with 235 more rows
```

Ad esempio, la parola più comune associata al sentimento da seguire "not" era "like", che normalmente avrebbe un punteggio (positivo) di 2.
Vale la pena chiedere quali parole hanno contribuito di più nella direzione "sbagliata". Per calcolare ciò, possiamo moltiplicare il loro punteggio per il numero di volte in cui appaiono (in modo che una parola con un punteggio di +3 che si verifica 10 volte abbia l'impatto di una parola con un punteggio di sentimento di +1 che si verifica 30 volte). Visualizziamo il risultato con un grafico a barre (Figura 4.2).

```
library(ggplot2)

not_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```

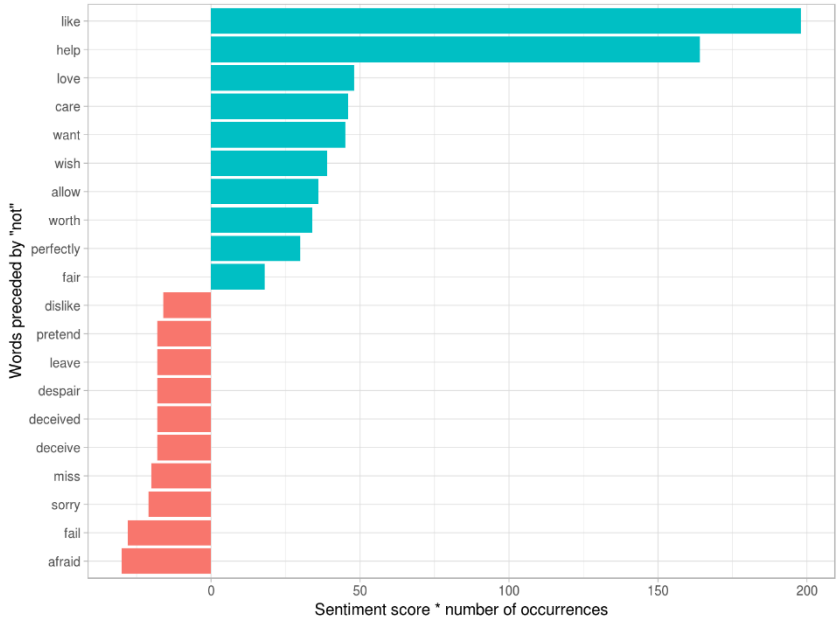


Figura 4.2: Le 20 parole precedute da "non" hanno avuto il maggior contributo ai punteggi del sentiment, sia in direzione positiva che negativa

I bigram "non mi piace" e "non aiutare" sono stati in gran parte le maggiori cause di errata identificazione, rendendo il testo molto più positivo di quanto non sia. Ma possiamo vedere frasi come "non aver paura" e "non fallire" a volte suggeriscono che il testo è più negativo di quello che è.

"Not" non è l'unico termine che fornisce un contesto per la seguente parola. Potremmo scegliere quattro parole comuni (o più) che annullano il termine successivo e utilizzare lo stesso approccio di unione e di conteggio per esaminarle tutte contemporaneamente.

```
negation_words <- c("not", "no", "never", "without")

negated_words <- bigrams_separated %>%
  filter(word1 %in% negation_words) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word1, word2, score, sort = TRUE) %>%
  ungroup()
```

Potremmo quindi visualizzare quali sono le parole più comuni per seguire ciascuna particolare negazione (Figura 4.3). Mentre "non mi piace" e "non aiutare" sono ancora i due esempi più comuni, possiamo anche vedere abbinamenti come "no great" e "never loved". Potremmo combinare questo con gli approcci nel Capitolo 2 per invertire i punteggi AFINN di ogni parola che segue una negazione. Questi sono solo alcuni esempi di come trovare parole consecutive può dare un contesto ai metodi di mining del testo.

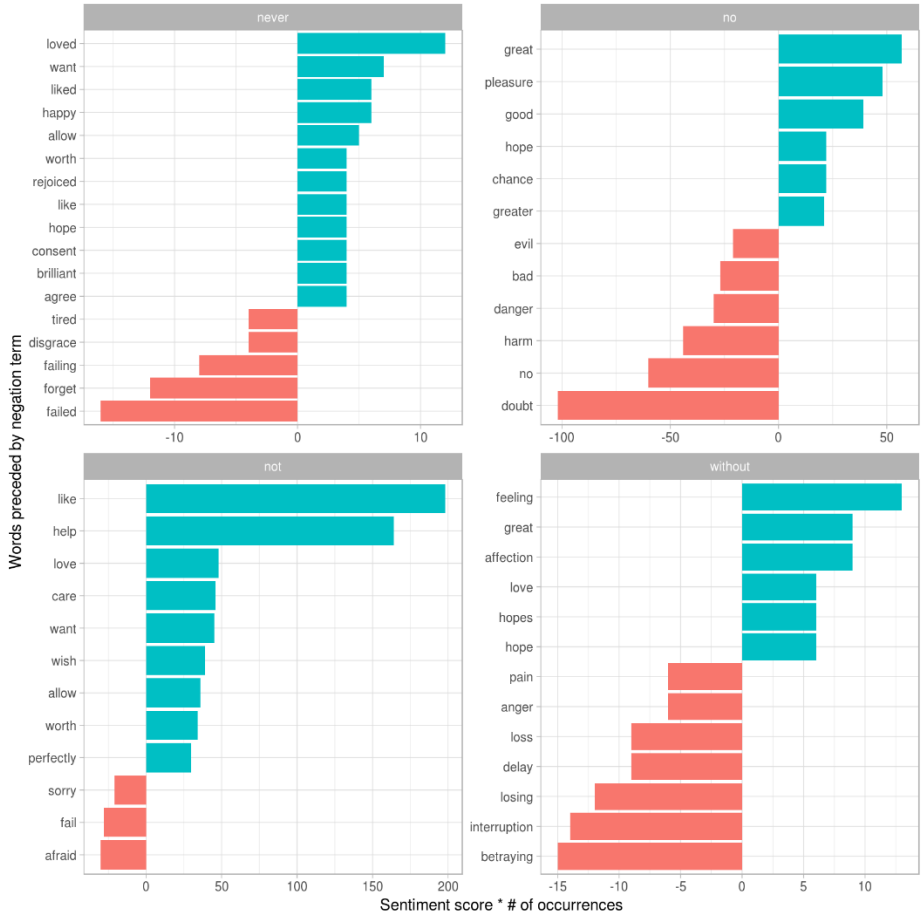


Figura 4.3: Le parole positive o negative più comuni per seguire negazioni come "mai", "no", "non" e "senza"

5.4.1.4 Visualizzare una rete di bigram con ggraph

Potremmo essere interessati a visualizzare tutte le relazioni tra le parole contemporaneamente, piuttosto che solo le poche in una volta. Come una visualizzazione comune, possiamo organizzare le parole in una rete, o "grafico". Qui ci riferiremo a un "**grafico**" non nel senso di una visualizzazione, ma come una combinazione di nodi connessi. Un grafico può essere costruito da un oggetto ordinato in quanto ha tre variabili:

- **da** : il nodo da cui proviene uno spigolo
- **a** : il nodo verso cui sta andando
- **peso** : un valore numerico associato a ciascun bordo

Il pacchetto **igraph** ha molte potenti funzioni per manipolare e analizzare le reti. Un modo per creare un oggetto igraph dai dati di ordinamento è la funzione `graph_from_data_frame` funzione che prende un dataframe di archi con colonne "from", "to" e gli attributi degli archi (in questo caso `n`):

```
library(igraph)

# original counts
bigram_counts
## # A tibble: 33,421 x 3
##    word1   word2         n
##    <chr>   <chr>     <int>
##  1 sir     thomas      287
##  2 miss    crawford    215
##  3 captain wentworth   170
##  4 miss    woodhouse   162
```

```
##  5 frank   churchill   132
##  6 lady    russell     118
##  7 lady    bertram     114
##  8 sir     walter      113
##  9 miss    fairfax     109
## 10 colonel brandon     108
## # ... with 33,411 more rows
```

```
# filter for only relatively common combinations
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph
```

```
## IGRAPH 536fefc DN-- 91 77 --
## + attr: name (v/c), n (e/n)
## + edges from 536fefc (vertex names):
##  [1] sir     ->thomas     miss    ->crawford   captain ->wentworth  miss
->woodhouse
##  [5] frank   ->churchill  lady    ->russell    lady    ->bertram    sir
->walter
##  [9] miss    ->fairfax    colonel ->brandon    miss    ->bates      lady
->catherine
## [13] sir     ->john       jane    ->fairfax    miss    ->tilney     lady
->middleton
## [17] miss    ->bingley    thousand->pounds     miss    ->dashwood   miss
->bennet
## [21] john    ->knightley  miss    ->morland    captain ->benwick    dear
->miss
## [25] miss    ->smith      miss    ->crawford's henry   ->crawford   miss
->elliot
## [29] dr      ->grant      miss    ->bertram    sir     ->thomas's   ten
->minutes
## + ... omitted several edges
```

igraph ha funzioni di plottaggio integrate, ma non sono ciò che il pacchetto è progettato per fare, quindi molti altri pacchetti hanno sviluppato metodi di visualizzazione per oggetti grafici. Raccomandiamo il pacchetto **ggraph**, perché implementa queste visualizzazioni in termini di grammatica della grafica, che conosciamo già da ggplot2.
Possiamo convertire un oggetto igraph in un `ggraph` con la funzione ggraph, dopo di che aggiungiamo dei livelli ad esso, proprio come i livelli sono aggiunti in ggplot2. Ad esempio, per un grafico di base è necessario aggiungere tre livelli: nodi, archi e testo.

```
library(ggraph)
set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

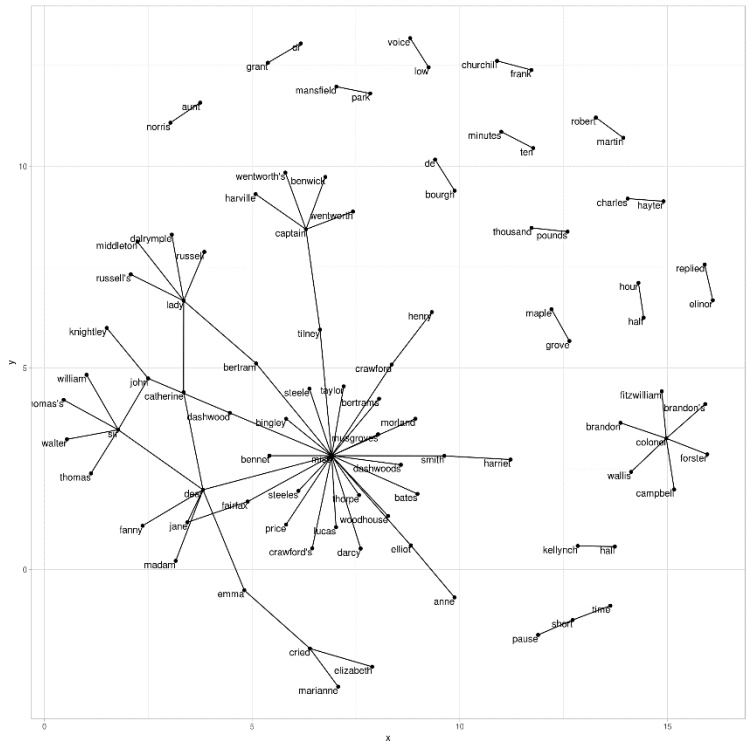


Figura 4.4: Bigram comuni nei romanzi di Jane Austen, che mostrano quelli che si sono verificati più di 20 volte e in cui nessuna parola era una parola d'arresto

Nella Figura 4.4, possiamo visualizzare alcuni dettagli della struttura del testo. Ad esempio, vediamo che saluti come "miss", "lady", "sir", "e" colonel "formano centri comuni di nodi, che sono spesso seguiti da nomi. Vediamo anche coppie o terzine lungo l'esterno che formano frasi brevi comuni ("mezz'ora", "mille sterline" o "breve tempo / pausa").

Concludiamo con alcune operazioni di lucidatura per ottenere un grafico migliore (Figura 4.5):

- Aggiungiamo l'estetica `edge_alpha` al livello link per rendere i collegamenti trasparenti in base a quanto comune o raro sia il bigram
- Aggiungiamo la direzionalità con una freccia, costruita usando `grid::arrow()`, inclusa un'opzione `end_cap` che dice alla freccia di finire prima di toccare il nodo
- Combiniamo le opzioni al livello nodo per rendere i nodi più attraenti (punti blu più grandi)
- Aggiungiamo un tema utile per tracciare reti, `theme_void()`

```
set.seed(2016)
a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```

Figura 4.5: Bigram comuni nei romanzi di Jane Austen, con alcune lucidature

Potrebbero essere necessari alcuni esperimenti con ggraph per ottenere le reti in un formato presentabile come questo, ma la struttura di rete è un modo utile e flessibile per visualizzare i dati di ordine relazionale.

Si noti che questa è una visualizzazione di una **catena di Markov**, un modello comune nell'elaborazione del testo. In una catena di Markov, ogni scelta di parole dipende solo dalla parola precedente. In questo caso, un generatore casuale che segue questo modello potrebbe sputare "caro", quindi "signore", quindi "william / walter / thomas / thomas's", seguendo ogni parola per le parole più comuni che lo seguono. Per rendere la visualizzazione interpretabile, abbiamo scelto di mostrare solo le più comuni connessioni parola per parola, ma si potrebbe immaginare un enorme grafico che rappresenta tutte le connessioni che si verificano nel testo.

5.4.1.5 Visualizzazione di bigram in altri testi

Abbiamo lavorato molto su come pulire e visualizzare i bigram su un set di dati di testo, quindi raccogliamolo in una funzione in modo da eseguirlo facilmente su altri set di dati di testo.

Per rendere più facile da usare le funzioni `count_bigrams()` e `visualize_bigrams()` da sole, abbiamo anche ricaricato i pacchetti necessari per loro.

```
library(dplyr)
library(tidyr)
library(tidytext)
library(ggplot2)
library(igraph)
```

```
library(ggraph)

count_bigrams <- function(dataset) {
  dataset %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
    separate(bigram, c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word,
           !word2 %in% stop_words$word) %>%
    count(word1, word2, sort = TRUE)
}

visualize_bigrams <- function(bigrams) {
  set.seed(2016)
  a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

  bigrams %>%
    graph_from_data_frame() %>%
    ggraph(layout = "fr") +
    geom_edge_link(aes(edge_alpha = n), show.legend = FALSE, arrow = a) +
    geom_node_point(color = "lightblue", size = 5) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()
}
```

A questo punto, potremmo visualizzare i bigram in altre opere, come la versione di Re Giacomo della Bibbia:

```
# the King James version is book 10 on Project Gutenberg:
library(gutenbergr)
kjv <- gutenberg_download(10)
```

```
library(stringr)

kjv_bigrams <- kjv %>%
  count_bigrams()

# filter out rare combinations, as well as digits
kjv_bigrams %>%
  filter(n > 40,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d")) %>%
  visualize_bigrams()
```

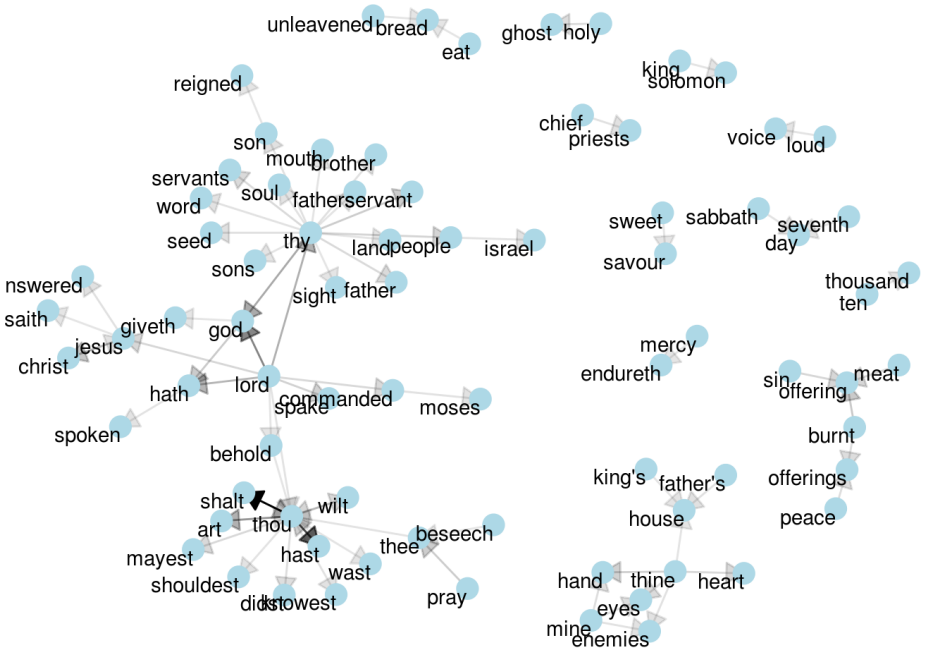


Figura 4.6: Grafico diretto dei bigram comuni nella Bibbia di Re Giacomo, che mostra quelli che si sono verificati più di 40 volte

La Figura 4.6 espone così un comune "programma" di linguaggio all'interno della Bibbia, particolarmente focalizzato intorno a "tuo" e "tu" (che potrebbe probabilmente essere considerato una parola d'ordine!) Puoi usare il pacchetto gutenbergr e queste funzioni `count_bigrams`/ `visualize_bigrams` per visualizzare i bigram in altri libri classici a cui sei interessato

5.4.2 Conteggio e correlazione di coppie di parole con il pacchetto widyr

Tokenizzare con n-grammi è un modo utile per esplorare coppie di parole adiacenti. Tuttavia, potremmo anche essere interessati a parole che tendono a coesistere all'interno di particolari documenti o capitoli particolari, anche se non si verificano uno accanto all'altro.

I tidy data sono una struttura utile per il confronto tra le variabili o il raggruppamento per righe, ma può essere difficile confrontare le righe: per esempio, contare il numero di volte in cui due parole compaiono nello stesso documento, o vedere come sono correlate. La maggior parte delle operazioni per la ricerca di conteggi o correlazioni a coppie ha bisogno di trasformare prima i dati in una matrice ampia.

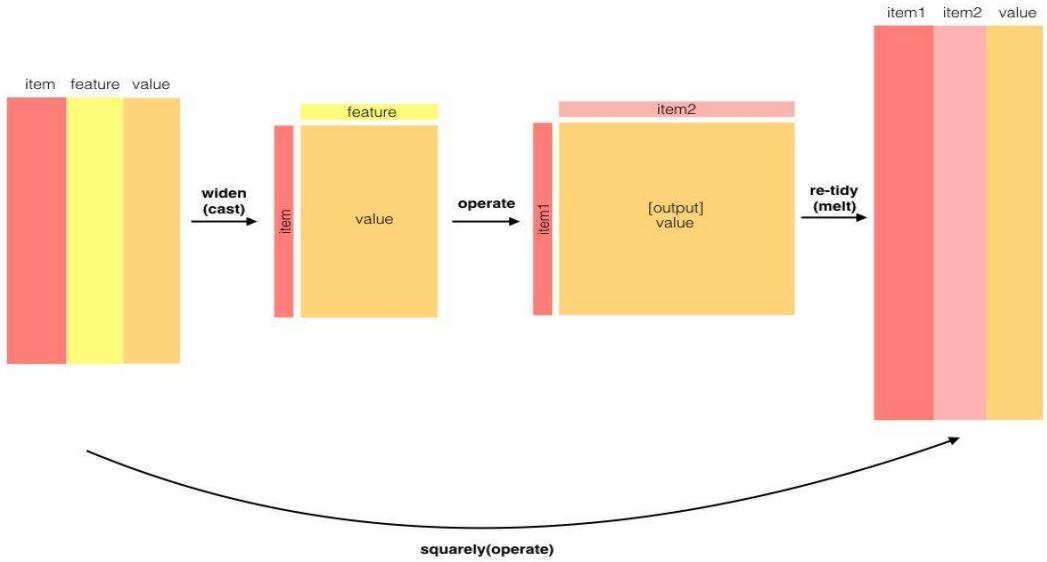


Figura 4.7: La filosofia alla base del pacchetto widyr, che può eseguire operazioni come il conteggio e la correlazione su coppie di valori in un dataset ordinato. Il pacchetto widyr dapprima "lancia" dataframe tidy in una matrice ampia, poi esegue un'operazione come una correlazione su di essa, quindi riordina il risultato.

Esamineremo alcuni dei modi in cui il testo ordinato può essere trasformato in una matrice ampia nel Capitolo 5 , ma in questo caso non è necessario. Il pacchetto widyr semplifica operazioni come il conteggio dei calcoli e le correlazioni, semplificando il modello di "ampliare i dati, eseguire un'operazione, quindi riordinare i dati" (Figura 4.7). Ci concentreremo su un insieme di funzioni che effettuano confronti a coppie tra gruppi di osservazioni (ad esempio, tra documenti o sezioni di testo).

5.4.2.1 Conteggio e correlazione tra sezioni

Considera il libro "Orgoglio e pregiudizio" diviso in sezioni di 10 righe, come abbiamo fatto (con sezioni più ampie) per l'analisi del sentimento nel Capitolo 2. Potremmo essere interessati a quali parole tendono ad apparire all'interno della stessa sezione.

```
austen_section_words <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  mutate(section = row_number() %/% 10) %>%
  filter(section > 0) %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word)

austen_section_words
```

```
## # A tibble: 37,240 x 3
##    book              section word
##    <fct>               <dbl> <chr>
##  1 Pride & Prejudice       1 truth
##  2 Pride & Prejudice       1 universally
##  3 Pride & Prejudice       1 acknowledged
##  4 Pride & Prejudice       1 single
```

```
##  5 Pride & Prejudice       1 possession
##  6 Pride & Prejudice       1 fortune
##  7 Pride & Prejudice       1 wife
##  8 Pride & Prejudice       1 feelings
##  9 Pride & Prejudice       1 views
## 10 Pride & Prejudice       1 entering
## # ... with 37,230 more rows
```

Una funzione utile di widyr è la funzione `pairwise_count`. Il prefisso `pairwise_` significa che risulterà in una riga per ogni coppia di parole nella variabile `word`. Questo ci consente di contare coppie di parole comuni che compaiono all'interno della stessa sezione:

```
library(widyr)

# count words co-occuring within sections
word_pairs <- austen_section_words %>%
  pairwise_count(word, section, sort = TRUE)

word_pairs
```

```
## # A tibble: 796,008 x 3
##    item1     item2         n
##    <chr>     <chr>     <dbl>
##  1 darcy     elizabeth   144
##  2 elizabeth darcy       144
##  3 miss      elizabeth   110
##  4 elizabeth miss        110
##  5 elizabeth jane        106
##  6 jane      elizabeth   106
##  7 miss      darcy        92
##  8 darcy     miss         92
##  9 elizabeth bingley      91
## 10 bingley   elizabeth    91
## # ... with 795,998 more rows
```

Si noti che mentre l'input aveva una riga per ogni coppia di un documento (una sezione di 10 righe) e una parola, l'output ha una riga per ogni coppia di parole. Questo è anche un formato tidy, ma di una struttura molto diversa che possiamo usare per rispondere a nuove domande.

Ad esempio, possiamo vedere che la coppia di parole più comune in una sezione è "Elizabeth" e "Darcy" (i due personaggi principali). Possiamo facilmente trovare le parole che si presentano più spesso con Darcy:

```
word_pairs %>%
  filter(item1 == "darcy")
```

```
## # A tibble: 2,930 x 3
##    item1 item2         n
##    <chr> <chr>     <dbl>
##  1 darcy elizabeth   144
##  2 darcy miss         92
##  3 darcy bingley      86
##  4 darcy jane         46
```

```
##  5 darcy bennet       45
##  6 darcy sister       45
##  7 darcy time         41
##  8 darcy lady         38
##  9 darcy friend       37
## 10 darcy wickham      37
## # ... with 2,920 more rows
```

5.4.2.2 Correlazione a coppie

In particolare, qui ci concentreremo sul coefficiente phi, una misura comune per la correlazione binaria. Il punto focale del coefficiente phi è quanto è più probabile che compaiono **sia** la parola X **che** Y, o che **nessuna delle due** fa, di quella che appare senza l'altra. Considera la seguente tabella:

	Has word Y	**No word Y**	**Total**
Has word X	n_{11}	n_{10}	n_1
No word X	n_{01}	n_{00}	n_0
Total	$n_{\cdot 1}$	$n_{\cdot 0}$	n

Ad esempio, n_{11} rappresenta il numero di documenti in cui compaiono sia la parola X che la parola Y, n_{00} il numero in cui nessuno dei due appare, e n_{10} e n_{01} i casi in cui uno appare senza l'altro. In termini di questa tabella, il coefficiente phi è:

$$\phi = n_{11}n_{00} - n_{10}n_{01} / \sqrt{n_1 \cdot n_0 \cdot n_{\cdot 0} n_{\cdot 1}}$$

Il coefficiente phi è equivalente alla correlazione di Pearson, che potresti aver sentito parlare altrove, quando è applicata ai dati binari). La funzione pairwise_cor () in widyr ci permette di trovare il coefficiente phi tra le parole in base alla frequenza con cui appaiono nella stessa sezione. La sua sintassi è simile a pairwise_count ().

Coppie come "Elizabeth" e "Darcy" sono le parole più comuni di co-occorrenza, ma ciò non è particolarmente significativo in quanto sono anche le parole individuali più comuni. Potremmo invece voler esaminare la **correlazione** tra le parole, che indica quanto spesso appaiono insieme relativamente alla frequenza con cui appaiono separatamente.
In particolare, qui ci concentreremo sul coefficiente phi , una misura comune per la correlazione binaria. Il punto focale del coefficiente phi è quanto è più probabile che compaiono sia la parola X che Y, o che nessuna delle due fa, di quella che appare senza l'altra.

Considera la seguente tabella:

	Ha la parola Y	Nessuna parola Y	Totale
Ha la parola X	n_{11}	n_{10}	$n_{1.}$
Nessuna parola X	n_{01}	n_{00}	$n_{0.}$
Totale	$n_{.1}$	$n_{.0}$	n

Ad esempio, che n_{11} rappresenta il numero di documenti in cui compaiono sia la parola X che la parola Y, n_{00} il numero in cui nessuno dei due appare e n_{10} e n_{01} i casi in cui uno appare senza l'altro. In termini di questa tabella, il coefficiente phi è:

$$\varphi = \frac{n_{11} n_{00} - n_{10} n_{01}}{\sqrt{n_{1.} n_{0.} n_{.0} n_{.1}}}$$

Il coefficiente phi è equivalente alla correlazione di Pearson, che potresti aver sentito parlare altrove, quando è applicata ai dati binari).

La funzione `pairwise_cor()` in widyr ci permette di trovare il coefficiente phi tra le parole in base alla frequenza con cui appaiono nella stessa sezione. La sua sintassi è simile a `pairwise_count()`.

```
# we need to filter for at least relatively common words first
word_cors <- austen_section_words %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, section, sort = TRUE)

word_cors
```

```
## # A tibble: 154,842 x 3
##    item1     item2     correlation
##    <chr>     <chr>           <dbl>
##  1 bourgh    de              0.951
##  2 de        bourgh          0.951
##  3 pounds    thousand        0.701
##  4 thousand  pounds          0.701
##  5 william   sir             0.664
##  6 sir       william         0.664
##  7 catherine lady            0.663
##  8 lady      catherine       0.663
##  9 forster   colonel         0.622
## 10 colonel   forster         0.622
## # ... with 154,832 more rows
```

Questo formato di output è utile per l'esplorazione. Ad esempio, potremmo trovare le parole più correlate con una parola come "sterline" usando `filter` un'operazione.

```
word_cors %>%
  filter(item1 == "pounds")
```

```
## # A tibble: 393 x 3
##    item1  item2     correlation
##    <chr>  <chr>           <dbl>
##  1 pounds thousand       0.701
##  2 pounds ten            0.231
##  3 pounds fortune        0.164
##  4 pounds settled        0.149
##  5 pounds wickham's      0.142
##  6 pounds children       0.129
##  7 pounds mother's       0.119
##  8 pounds believed       0.0932
##  9 pounds estate         0.0890
## 10 pounds ready          0.0860
## # ... with 383 more rows
```

Questo ci permette di selezionare parole particolarmente interessanti e trovare le altre parole più associate a loro (Figura 4.8).

```
word_cors %>%
  filter(item1 %in% c("elizabeth", "pounds", "married", "pride")) %>%
  group_by(item1) %>%
  top_n(6) %>%
  ungroup() %>%
  mutate(item2 = reorder(item2, correlation)) %>%
  ggplot(aes(item2, correlation)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ item1, scales = "free") +
  coord_flip()
```

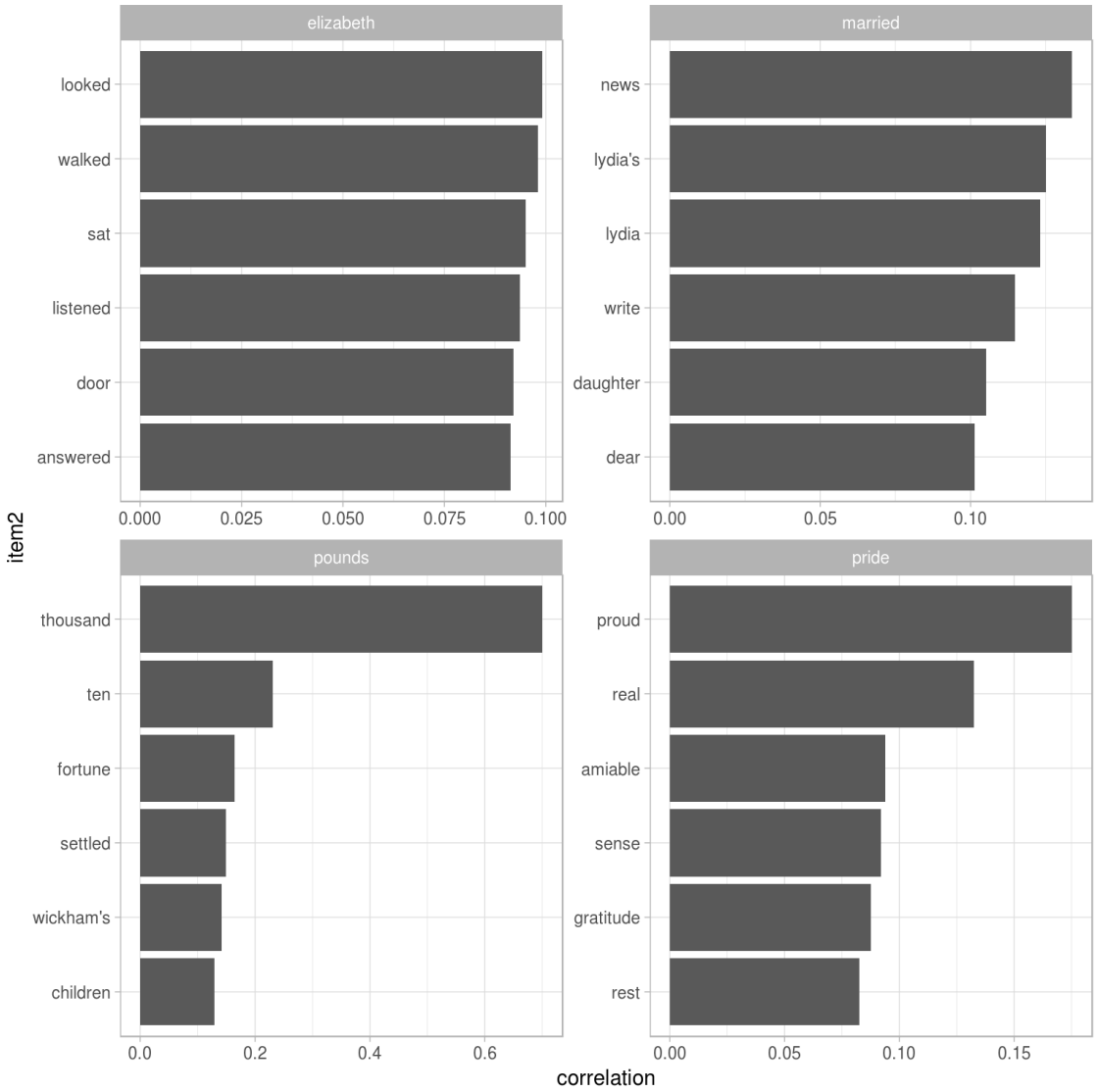


Figura 4.8: Parole di orgoglio e pregiudizio che erano maggiormente correlate con "elizabeth", "pounds", "married" e "orgoglio"

Proprio come abbiamo usato ggraph per visualizzare i bigram, possiamo usarlo per visualizzare le correlazioni e i gruppi di parole che sono stati trovati dal pacchetto widyr (Figura 4.9).

```
set.seed(2016)

word_cors %>%
  filter(correlation > .15) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

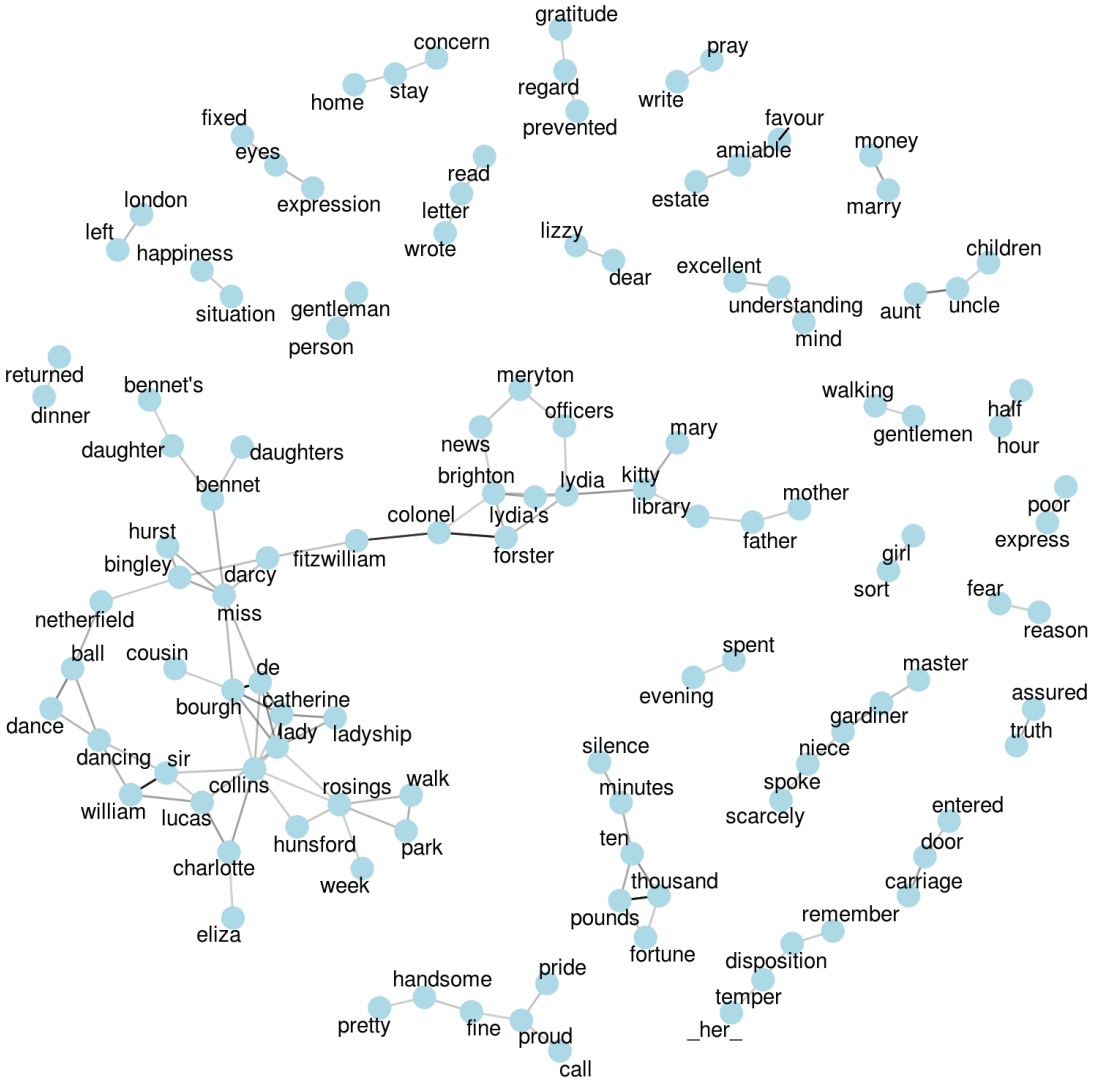


Figura 4.9: Coppie di parole in Orgoglio e Pregiudizio che mostrano almeno una correlazione 0.1 tra la stessa sezione di 10 righe

Nota che a differenza dell'analisi del bigram, le relazioni qui sono simmetriche, piuttosto che direzionali (non ci sono frecce). Possiamo anche vedere che mentre gli accoppiamenti di nomi e titoli che hanno dominato le coppie di bigram sono comuni, come "colonnello / fitzwilliam", possiamo anche vedere coppie di parole che appaiono vicine l'una all'altra, come "camminare" e "parcheggiare", o "danza" e "palla".

5.4.3 Riepilogo

Questo capitolo ha mostrato come l'approccio al tidy text sia utile non solo per analizzare singole parole, ma anche per esplorare le relazioni e le connessioni tra le parole. Tali relazioni possono coinvolgere n-grammi, che ci permettono di vedere quali parole tendono ad apparire dopo le altre, o co-occorrenze e correlazioni, per parole che appaiono in prossimità l'una dell'altra. Questo capitolo ha anche dimostrato il pacchetto ggraph per la visualizzazione di entrambi questi tipi di relazioni come reti. Queste visualizzazioni di rete sono uno strumento flessibile per esplorare le relazioni e svolgeranno un ruolo importante negli studi di casi nei capitoli successivi.

5 Conversione da e verso formati non ordinati

Nei capitoli precedenti, abbiamo analizzato il testo disposto nel formato di testo ordinato: una tabella con un token-per-document-per-riga, come è costruito dalla funzione `unnest_tokens()`. Questo ci consente di utilizzare la famosa suite di strumenti di ordinamento come dplyr, tidyr e ggplot2 per esplorare e visualizzare i dati di testo. Abbiamo dimostrato che molte analisi di testo informativo possono essere eseguite utilizzando questi strumenti.

Tuttavia, la maggior parte degli strumenti R esistenti per l'elaborazione del linguaggio naturale, oltre al pacchetto tidytext, non sono compatibili con questo formato. La CRAN per l'elaborazione del linguaggio naturale elenca una vasta selezione pacchetti di R che prendono altre strutture di input e forniscono output non ordinati. Questi pacchetti sono molto utili nelle applicazioni di text mining e molti dataset di testo esistenti sono strutturati in base a questi formati.

Lo scienziato informatico Hal Abelson ha osservato che "Non importa quanto complesse e raffinate siano le singole operazioni, spesso è la qualità della colla che determina più direttamente la potenza del sistema" (Abelson 2008) . In questo spirito, questo capitolo si discuterà la "colla" che collega il formato di testo ordinato con altri importanti pacchetti e strutture dati, consentendo di fare affidamento su entrambi i pacchetti di text mining esistenti e sulla suite di strumenti ordinati per eseguire l'analisi.

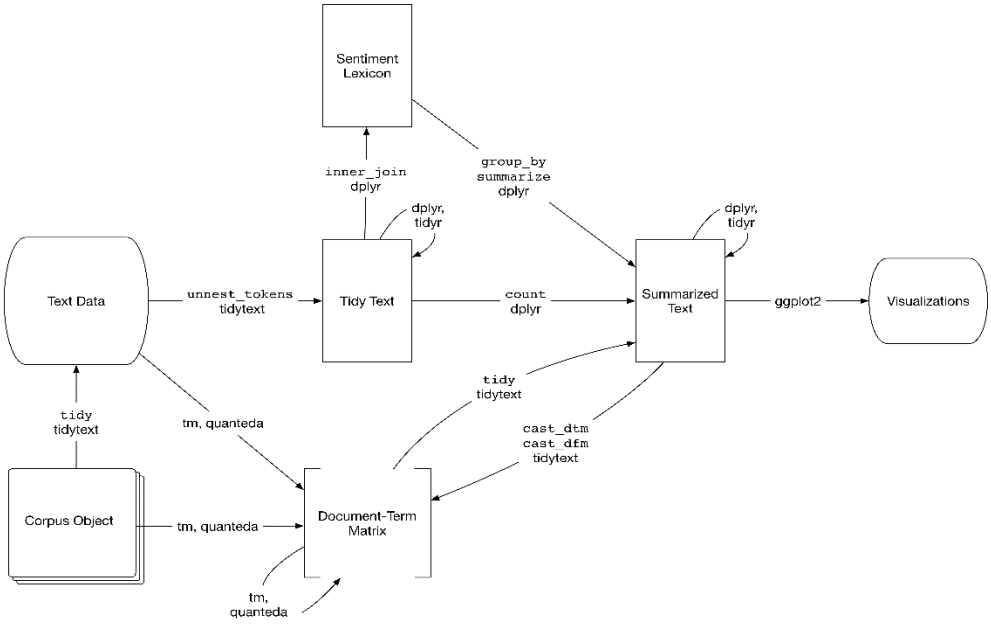


Figura 5.1: Un diagramma di flusso di una tipica analisi del testo che combina il tidytext con altri strumenti e formati di dati, in particolare i pacchetti tm o quanteda. Questo capitolo mostra come convertire avanti e indietro tra matrici di documenti e ordinamenti di dati, nonché la conversione da un oggetto Corpus a una text data frame.

La Figura 5.1 illustra come un'analisi potrebbe passare da strutture e strumenti di dati ordinati e non ordinati. Questo capitolo si concentrerà sul processo di riordino delle matrici a termine del documento e sul cast di un frame ordinato in una matrice sparsa. Esploreremo anche come riordinare gli oggetti Corpus, che combinano il testo non elaborato con i metadati del documento, in frame di dati di testo, portando a un case study sull'ingestione e l'analisi di articoli finanziari.

5.5.1 Riordinare una matrice di termini documento

L'analisi del testo tipicamente non usa i dataframe ma usa la **Document-term-matrix** (DTM). Questa matrice sulle righe ha i documenti, sulle colonne ha le parole e avrò un numero r che mi dice quante volte la parola occorre nel documento. Tipicamente viene utilizzata per fare l'analisi del testo. È necessario avere delle funzioni che ci facciano passare dalla rappresentazione del testo in data frame ad una rappresentazione del testo in DTM e viceversa.

- FUNZIONE tidy() turns a document-term matrix into a tidy data frame
- FUNZIONE cast() turns a tidy one-term-per-row data frame into a matrix

Una delle strutture più comuni con cui lavorano i pacchetti di mining del testo è la **matrice dei termini del documento** (o **DTM**). Questa è una matrice in cui:

- ogni riga rappresenta un documento (come un libro o un articolo),
- ogni colonna rappresenta un termine, e
- ogni valore (tipicamente) contiene il numero di aspetti di quel termine in quel documento.

Poiché la maggior parte degli accoppiamenti di documento e termine non si verificano (hanno il valore zero), i DTM sono solitamente implementati come matrici sparse (0 fuori da diagonale). Questi oggetti possono essere trattati come se fossero matrici (ad esempio, l'accesso a particolari righe e colonne), ma sono memorizzati in un formato più efficiente. Discuteremo varie implementazioni di queste matrici in questo capitolo.

Gli oggetti DTM non possono essere utilizzati direttamente con gli tidy tools, così come i tidy data frame non possono essere utilizzati come input per la maggior parte dei pacchetti di text mining. Pertanto, il pacchetto tidytext fornisce due funzioni che convertono i due formati.

- `tidy()` trasforma una matrice di termini del documento in una tidy data frame. Questo verbo deriva dal pacchetto broom, che fornisce funzioni di riordino simili per molti modelli e oggetti statistici.
- `cast()` trasforma un data frame tidy con un singolo termine per riga in una matrice. tidytext fornisce tre varianti di questo verbo, ciascuna convertita in un diverso tipo di matrice: `cast_sparse()` (convertendosi in una matrice sparsa dal pacchetto Matrix), `cast_dtm()` (convertendosi in un oggetto `DocumentTermMatrix` da tm) e `cast_dfm()` (convertendosi in un oggetto `dfm` da quanteda).

Come mostrato nella Figura 5.1, un DTM è in genere paragonabile a un frame di dati ordinato dopo a `count` o a `group_by`/ `summarize` che contiene conteggi o un'altra statistica per ciascuna combinazione di un termine e di un documento.

5.1.1 Riordinare oggetti DocumentTermMatrix

Forse l'implementazione più diffusa dei DTM in R è la classe `DocumentTermMatrix` nel pacchetto tm. Molti set di dati di estrazione di testo disponibili sono forniti in questo formato. Ad esempio, si consideri la raccolta di articoli di giornale della Associated Press inclusi nel pacchetto topicmodels.

```
library(tm)

data("AssociatedPress", package = "topicmodels")
```

```
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Vediamo che questo set di dati contiene documenti (ognuno di essi un articolo AP) e termini (parole distinte). Si noti che questo DTM è al 99% scarso (il 99% delle coppie di parole-documento è zero). Possiamo accedere ai termini nel documento con la funzione `Terms()`.

```
terms <- Terms(AssociatedPress)
head(terms)
## [1] "aaron"     "abandon"    "abandoned"  "abandoning" "abbott"     "abboud"
```

Se volessimo analizzare questi dati con gli strumenti ordinati, dovremmo prima trasformarli in un frame di dati con un token-per-document-per-row. Il pacchetto broom ha introdotto il verbo tidy(), che prende un oggetto non ordinato e lo trasforma in una data frame ordinato. Il pacchetto tidytext implementa questo metodo per gli oggetti `DocumentTermMatrix`.

```
library(dplyr)
library(tidytext)

ap_td <- tidy(AssociatedPress)
ap_td
```

```
## # A tibble: 302,031 x 3
##    document term       count
##       <int> <chr>      <dbl>
##  1        1 adding         1
##  2        1 adult          2
##  3        1 ago            1
##  4        1 alcohol        1
##  5        1 allegedly      1
##  6        1 allen          1
##  7        1 apparently     2
##  8        1 appeared       1
##  9        1 arrested       1
## 10        1 assault        1
## # ... with 302,021 more rows
```

Si noti che ora abbiamo tre colonne tidy `tbl_df`, con le variabili `document`, `term` e `count`. Questa operazione di riordino `melt()` è simile alla funzione del pacchetto reshape2 per le matrici non sparse.

Si noti che solo i valori diversi da zero sono inclusi nell'output ordinato: il documento 1 include termini come "aggiunta" e "adulto", ma non "aaron" o "abbandon". Ciò significa che la versione riordinata non ha righe dove `count` è zero.

Come abbiamo visto nei capitoli precedenti, questo modulo è utile per l'analisi con i pacchetti dplyr, tidytext e ggplot2. Ad esempio, è possibile eseguire analisi del sentimento su questi articoli di giornale con l'approccio descritto nel Capitolo 2.

```
ap_sentiments <- ap_td %>%
  inner_join(get_sentiments("bing"), by = c(term = "word"))

ap_sentiments
```

```
# A tibble: 30,094 x 4
##    document term    count sentiment
##       <int> <chr>   <dbl> <chr>
##  1        1 assault     1 negative
##  2        1 complex     1 negative
##  3        1 death       1 negative
##  4        1 died        1 negative
##  5        1 good        2 positive
##  6        1 illness     1 negative
##  7        1 killed      2 negative
##  8        1 like        2 positive
##  9        1 liked       1 positive
## 10        1 miracle     1 positive
## # ... with 30,084 more rows
```

Questo ci permetterebbe di visualizzare quali parole degli articoli AP hanno più spesso contribuito a sentimenti positivi o negativi, come mostrato nella Figura 5.2. Possiamo vedere che le parole positive più comuni includono "like", "work", "support" e "good", mentre le parole più negative includono "dead", "death" e "vice". (L'inclusione di "vice" come termine negativo è probabilmente un errore da parte dell'algoritmo, dal momento che solitamente si riferisce al "vicepresidente").6

```
library(ggplot2)

ap_sentiments %>%
  count(sentiment, term, wt = count) %>%
  ungroup() %>%
  filter(n >= 200) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(term = reorder(term, n)) %>%
  ggplot(aes(term, n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  ylab("Contribution to sentiment") +
  coord_flip()
```

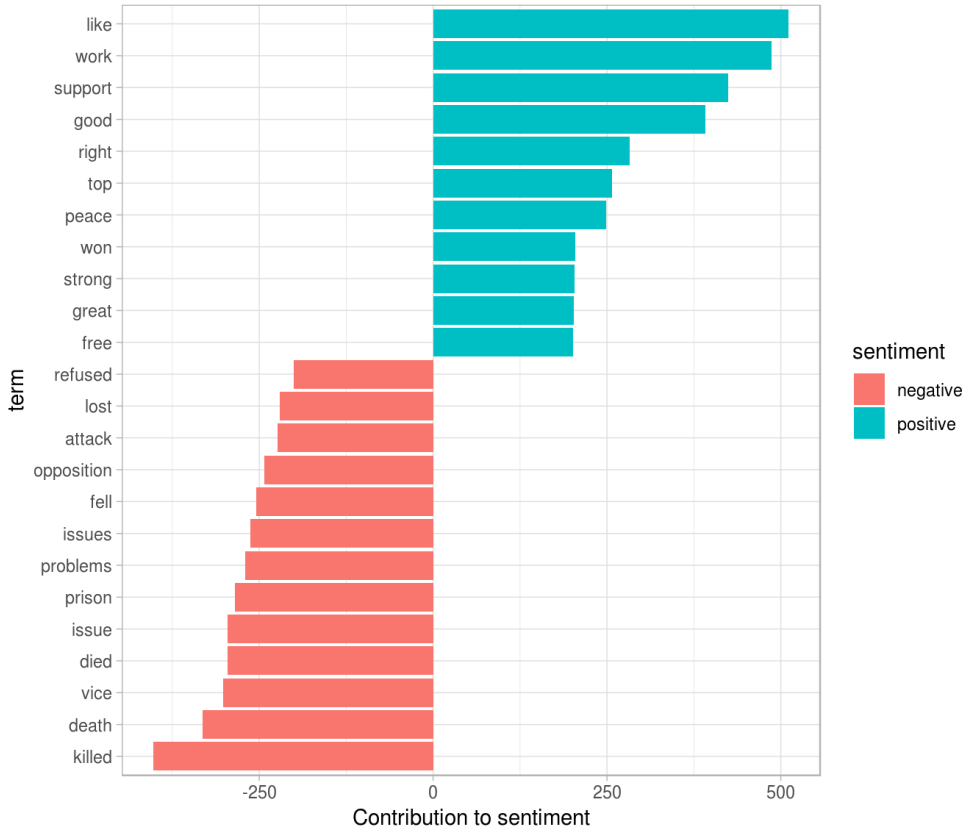


Figura 5.2: Parole di articoli di AP con il maggior contributo a sentimenti positivi o negativi, usando il lessico del sentimento di Bing.

5.1.2 Riordinare oggetti dfm

Altri pacchetti di mining di testo forniscono implementazioni alternative di document-term matrices , come la classe `dfm` (matrice dellecaratteristiche del documento) del pacchetto quanteda. Ad esempio, il pacchetto Quanteda viene fornito con un corpus di discorsi di inaugurazione presidenziale, che possono essere convertiti in una appropriata funzione dfm.

```
data("data_corpus_inaugural", package = "quanteda")
inaug_dfm <- quanteda::dfm(data_corpus_inaugural, verbose = FALSE)

inaug_dfm

## Document-feature matrix of: 58 documents, 9,357 features (91.8% sparse).
```

Il metodo `tidy` funziona anche su queste matrici di documenti, trasformandole in una tabella di un token per documento per riga:

```
inaug_td <- tidy(inaug_dfm)
inaug_td

## # A tibble: 44,709 x 3
##    document        term            count
##    <chr>           <chr>           <dbl>
##  1 1789-Washington fellow-citizens     1
##  2 1797-Adams      fellow-citizens     3
##  3 1801-Jefferson  fellow-citizens     2
```

```
##  4 1809-Madison    fellow-citizens     1
##  5 1813-Madison    fellow-citizens     1
##  6 1817-Monroe     fellow-citizens     5
##  7 1821-Monroe     fellow-citizens     1
##  8 1841-Harrison   fellow-citizens    11
##  9 1845-Polk       fellow-citizens     1
## 10 1849-Taylor     fellow-citizens     1
## # ... with 44,699 more rows
```

Potremmo essere interessati a trovare le parole più specifiche per ciascuno dei discorsi inaugurali. Questo potrebbe essere quantificato calcolando il tf-idf di ogni coppia di termini-discorso usando la funzione `bind_tf_idf()`, come descritto nel Capitolo 3 .

```
inaug_tf_idf <- inaug_td %>%
  bind_tf_idf(term, document, count) %>%
  arrange(desc(tf_idf))

inaug_tf_idf
```

```
## # A tibble: 44,709 x 6
##    document        term        count      tf   idf tf_idf
##    <chr>           <chr>       <dbl>   <dbl> <dbl>  <dbl>
##  1 1793-Washington arrive          1 0.00680  4.06 0.0276
##  2 1793-Washington upbraidings     1 0.00680  4.06 0.0276
##  3 1793-Washington violated        1 0.00680  3.37 0.0229
##  4 1793-Washington willingly       1 0.00680  3.37 0.0229
##  5 1793-Washington incurring       1 0.00680  3.37 0.0229
##  6 1793-Washington previous        1 0.00680  2.96 0.0201
##  7 1793-Washington knowingly       1 0.00680  2.96 0.0201
##  8 1793-Washington injunctions     1 0.00680  2.96 0.0201
##  9 1793-Washington witnesses       1 0.00680  2.96 0.0201
## 10 1793-Washington besides         1 0.00680  2.67 0.0182
## # ... with 44,699 more rows
```

Potremmo usare questi dati per scegliere quattro indirizzi inaugurali degni di nota (dai presidenti Lincoln, Roosevelt, Kennedy e Obama) e visualizzare le parole più specifiche per ciascun discorso, come mostrato nella Figura 5.3 .

Figura 5.3: i termini con il più alto tf-idf da ciascuno dei quattro indirizzi inaugurali selezionati. Nota che il tokenizer di quanteda include il '?' segno di punteggiatura come un termine, anche se i testi che abbiamo tokenizzato noi stessi con unnest_tokens non lo fanno.

Come un altro esempio di visualizzazione possibile con dati ordinati, potremmo estrarre l'anno dal nome di ogni documento e calcolare il numero totale di parole all'interno di ogni anno.

Nota che abbiamo usato la funzione `complete()` di tidyr per includere gli zeri (casi in cui una parola non appare in un documento) nella tabella.

```
library(tidyr)

year_term_counts <- inaug_td %>%
  extract(document, "year", "(\\d+)", convert = TRUE) %>%
```

```
  complete(year, term, fill = list(count = 0)) %>%
  group_by(year) %>%
  mutate(year_total = sum(count))
```

Questo ci permette di scegliere diverse parole e visualizzare come sono cambiate in frequenza nel tempo, come mostrato in 5.4. Possiamo vedere che nel corso del tempo, i presidenti americani hanno avuto meno probabilità di riferirsi al paese come "Unione" e più probabilmente di riferirsi a "America". Divennero anche meno propensi a parlare della "costituzione" e dei "paesi stranieri", e più probabilmente di menzionare "libertà" e "Dio".

```
year_term_counts %>%
  filter(term %in% c("god", "america", "foreign", "union", "constitution",
"freedom")) %>%
  ggplot(aes(year, count / year_total)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ term, scales = "free_y") +
  scale_y_continuous(labels = scales::percent_format()) +
  ylab("% frequency of word in inaugural address")
```

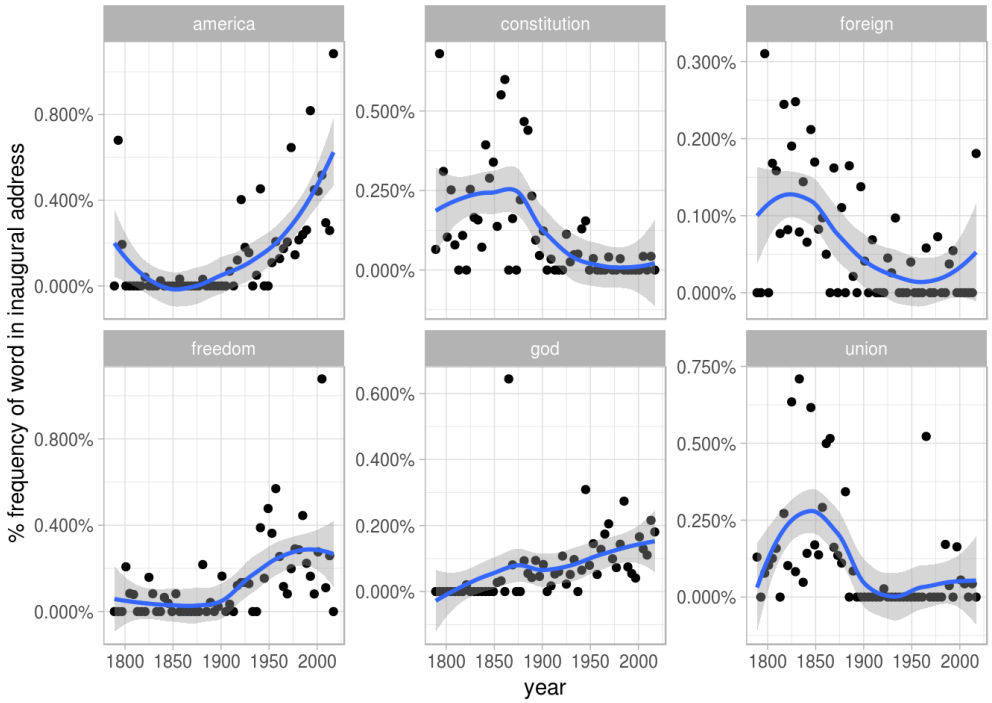


Figura 5.4: Variazioni della frequenza delle parole nel tempo all'interno degli discorsi inaugurali presidenziali, per sei termini selezionati

Questi esempi mostrano come utilizzare tidytext e la relativa suite di strumenti di ordinamento per analizzare le origini anche se la loro origine non era in un formato tidy.

5.2 Lancio di dati di testo ordinati in una matrice

Proprio come alcuni pacchetti di text mining esistenti forniscono matrici document-term come dati di esempio o output, alcuni algoritmi si aspettano tali matrici come input. Pertanto, tidytext fornisce i verbi `cast_` per la conversione da una forma tidy a queste matrici.

Ad esempio, potremmo prendere il set di dati dell'AP ordinato e ricollocarlo in una matrice di termini del documento usando la funzione `cast_dtm()`.

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Allo stesso modo, potremmo lanciare il tavolo in un dfmoggetto dal dfm di quanteda con `cast_dfm()`.

```
ap_td %>%
  cast_dfm(document, term, count)

## Document-feature matrix of: 2,246 documents, 10,473 features
(98.7% sparse).
```

Alcuni strumenti richiedono semplicemente una matrice sparsa:

```
ibrary(Matrix)

# cast into a Matrix object
m <- ap_td %>%
  cast_sparse(document, term, count)

class(m)
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"

dim(m)
## [1]  2246 10473
```

Questo tipo di conversione potrebbe facilmente essere fatto da qualsiasi delle strutture di testo ridy che abbiamo utilizzato finora in questo libro. Ad esempio, potremmo creare un DTM dei libri di Jane Austen in poche righe di codice.

```
library(janeaustenr)

austen_dtm <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  cast_dtm(book, word, n)

austen_dtm
```

```
## <<DocumentTermMatrix (documents: 6, terms: 14520)>>
## Non-/sparse entries: 40379/46741
## Sparsity           : 54%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

Questo processo di fusione consente la lettura, il filtraggio e l'elaborazione da eseguire utilizzando dplyr e altri strumenti di ordinamento, dopo i quali i dati possono essere convertiti in una matrice di termini documento per applicazioni di apprendimento automatico. Nel Capitolo 6 esamineremo alcuni esempi in cui un set di dati di testo ordinato deve essere convertito in una DocumentTermMatrix per l'elaborazione.

5.3 Riordinare gli oggetti corpus con i metadati

Alcune strutture di dati sono progettate per archiviare raccolte di documenti *prima* della tokenizzazione, spesso definite "corpus". Un esempio comune sono gli oggetti `Corpus` del pacchetto tm. Questi memorizzano il testo accanto ai metadati , che possono includere un ID, data / ora, titolo o lingua per ciascun documento.

Ad esempio, il pacchetto tm viene fornito con il corpus `acq`, contenente 50 articoli dal servizio di notizie Reuters.

```
data("acq")
acq
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 50
```

```
# first document
acq[[1]]
## <<PlainTextDocument>>
## Metadata:  15
## Content:  chars: 1287
```

Un oggetto corpus è strutturato come un elenco, con ciascun elemento contenente sia testo che metadati (consultare la documentazione di TM per ulteriori informazioni su come lavorare con i documenti Corpus). Questo è un metodo di archiviazione flessibile per i documenti, ma non si presta all'elaborazione con strumenti ordinati.

Possiamo quindi utilizzare il metodo `tidy()` per costruire una tabella con una riga per documento, inclusi i metadati (come `id` e `datetimestamp`) come colonne accanto a text.

```
acq_td <- tidy(acq)
acq_td
## # A tibble: 50 x 16
##    author  datetimestamp       description heading  id    language origin
topics lewissplit cgisplit
##    <chr>   <dttm>              <chr>       <chr>    <chr> <chr>    <chr>
<chr>  <chr>      <chr>
##  1 <NA>    1987-02-26 15:18:06 ""          COMPUTE… 10    en       Reute…
YES    TRAIN      TRAININ…
##  2 <NA>    1987-02-26 15:19:15 ""          OHIO MA… 12    en       Reute…
YES    TRAIN      TRAININ…
##  3 <NA>    1987-02-26 15:49:56 ""          MCLEAN'… 44    en       Reute…
YES    TRAIN      TRAININ…
##  4 By Cal… 1987-02-26 15:51:17 ""          CHEMLAW… 45    en       Reute…
YES    TRAIN      TRAININ…
##  5 <NA>    1987-02-26 16:08:33 ""          <COFAB … 68    en       Reute…
YES    TRAIN      TRAININ…
```

```
##  6 <NA>    1987-02-26 16:32:37 ""          INVESTM… 96    en       Reute…
YES    TRAIN      TRAININ…
##  7 By Pat… 1987-02-26 16:43:13 ""          AMERICA… 110   en       Reute…
YES    TRAIN      TRAININ…
##  8 <NA>    1987-02-26 16:59:25 ""          HONG KO… 125   en       Reute…
YES    TRAIN      TRAININ…
##  9 <NA>    1987-02-26 17:01:28 ""          LIEBERT… 128   en       Reute…
YES    TRAIN      TRAININ…
## 10 <NA>    1987-02-26 17:08:27 ""          GULF AP… 134   en       Reute…
YES    TRAIN      TRAININ…
##    oldid places   people orgs  exchanges text
##    <chr> <list>   <lgl>  <lgl> <lgl>     <chr>
##  1 5553  <chr [1… NA     NA    NA        "Computer Terminal Systems Inc
said\nit has completed the …
##  2 5555  <chr [1… NA     NA    NA        "Ohio Mattress Co said its
first\nquarter, ending February…
##  3 5587  <chr [1… NA     NA    NA        "McLean Industries Inc's
United\nStates Lines Inc subsidia…
##  4 5588  <chr [1… NA     NA    NA        "ChemLawn Corp <CHEM> could
attract a\nhigher bid than the…
##  5 5611  <chr [1… NA     NA    NA        "CoFAB Inc said it acquired
<Gulfex Inc>,\na Houston-based…
##  6 5639  <chr [1… NA     NA    NA        "A group of affiliated New
York\ninvestment firms said the…
##  7 5653  <chr [1… NA     NA    NA        "American Express Co remained
silent on\nmarket rumors it …
##  8 5668  <chr [1… NA     NA    NA        "Industrial Equity (Pacific) Ltd,
a\nHong Kong investment …
##  9 5671  <chr [1… NA     NA    NA        "Liebert Corp said its
shareholders\napproved the merger o…
## 10 5677  <chr [1… NA     NA    NA        "Gulf Applied Technologies Inc
said it\nsold its subsidiar…
## # ... with 40 more rows
```

Questo può quindi essere utilizzato unnest_tokens(), ad esempio, per trovare le parole più comuni tra gli articoli di 50 Reuters o quelli più specifici di ciascun articolo.

```
acq_tokens <- acq_td %>%
  select(-places) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")

# most common words
acq_tokens %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 1,566 x 2
##    word         n
##    <chr>    <int>
##  1 dlrs       100
##  2 pct         70
##  3 mln         65
##  4 company     63
```

```
##  5 shares      52
##  6 reuter      50
##  7 stock       46
##  8 offer       34
##  9 share       34
## 10 american    28
## # ... with 1,556 more rows
```

```
# tf-idf
acq_tokens %>%
  count(id, word) %>%
  bind_tf_idf(word, id, n) %>%
  arrange(desc(tf_idf))

## # A tibble: 2,853 x 6
##    id    word         n     tf   idf tf_idf
##    <chr> <chr>    <int>  <dbl> <dbl>  <dbl>
##  1 186   groupe       2 0.133   3.91  0.522
##  2 128   liebert      3 0.130   3.91  0.510
##  3 474   esselte      5 0.109   3.91  0.425
##  4 371   burdett      6 0.103   3.91  0.405
##  5 442   hazleton     4 0.103   3.91  0.401
##  6 199   circuit      5 0.102   3.91  0.399
##  7 162   suffield     2 0.1     3.91  0.391
##  8 498   west         3 0.1     3.91  0.391
##  9 441   rmj          8 0.121   3.22  0.390
## 10 467   nursery      3 0.0968  3.91  0.379
## # ... with 2,843 more rows
```

5.3.1 Esempio: articoli finanziari minerari

Gli oggetti `Corpus` sono un formato di output comune per i pacchetti di importazione dei dati, il che significa che la funzione `tidy()` ci consente di accedere ad un'ampia varietà di dati di testo. Un esempio è tm.plugin.webmining , che si collega ai feed online per recuperare articoli di notizie basati su una parola chiave. Ad esempio, l'esecuzione WebCorpus(GoogleFinanceSource("NASDAQ:MSFT")) ci consente di recuperare i 20 articoli più recenti relativi al titolo Microsoft (MSFT).

Qui recupereremo articoli recenti relativi a nove principali titoli tecnologici: Microsoft, Apple, Google, Amazon, Facebook, Twitter, IBM, Yahoo e Netflix.

Questi risultati sono stati scaricati a gennaio 2017, quando questo capitolo è stato scritto, ma troverai sicuramente risultati diversi se lo avessi eseguito da solo. Si noti che questo codice richiede diversi minuti per essere eseguito.

```
library(tm.plugin.webmining)
library(purrr)

company <- c("Microsoft", "Apple", "Google", "Amazon", "Facebook",
             "Twitter", "IBM", "Yahoo", "Netflix")
symbol <- c("MSFT", "AAPL", "GOOG", "AMZN", "FB", "TWTR", "IBM", "YHOO",
"NFLX")
```

```
download_articles <- function(symbol) {
  WebCorpus(GoogleFinanceSource(paste0("NASDAQ:", symbol)))
}

stock_articles <- data_frame(company = company,
                             symbol = symbol) %>%
  mutate(corpus = map(symbol, download_articles))
```

Questo usa la funzione `map()` dal pacchetto purrr, che applica una funzione a ciascun elemento in `symbol` per creare un elenco, che memorizziamo nella corpuscolonna dell'elenco.

```
stock_articles

## # A tibble: 9 x 3
##   company   symbol corpus
##   <chr>     <chr>  <list>
## 1 Microsoft MSFT   <S3: WebCorpus>
## 2 Apple     AAPL   <S3: WebCorpus>
## 3 Google    GOOG   <S3: WebCorpus>
## 4 Amazon    AMZN   <S3: WebCorpus>
## 5 Facebook  FB     <S3: WebCorpus>
## 6 Twitter   TWTR   <S3: WebCorpus>
## 7 IBM       IBM    <S3: WebCorpus>
## 8 Yahoo     YHOO   <S3: WebCorpus>
## 9 Netflix   NFLX   <S3: WebCorpus>
```

Ciascuno degli elementi nella corpuscolonna elenco è un oggetto `WebCorpus`, che è un caso speciale di un corpus come `acq`. Possiamo quindi trasformare ciascuno in un frame di dati usando la funzione `tidy()`, non più con quello di tidyr `unnest()`, quindi con un tokenizzare la textcolonna dei singoli articoli usando `unnest_tokens()`.

```
stock_tokens <- stock_articles %>%
  unnest(map(corpus, tidy)) %>%
  unnest_tokens(word, text) %>%
  select(company, datetimestamp, word, id, heading)

stock_tokens
```

```
## # A tibble: 105,054 x 5
##    company   datetimestamp       word        id
##    <chr>     <dttm>              <chr>       <chr>
##  1 Microsoft 2017-01-17 12:07:24 microsoft
tag:finance.google.com,cluster:52779347599411
##  2 Microsoft 2017-01-17 12:07:24 corporation
tag:finance.google.com,cluster:52779347599411
##  3 Microsoft 2017-01-17 12:07:24 data
tag:finance.google.com,cluster:52779347599411
##  4 Microsoft 2017-01-17 12:07:24 privacy
tag:finance.google.com,cluster:52779347599411
##  5 Microsoft 2017-01-17 12:07:24 could
tag:finance.google.com,cluster:52779347599411
```

```
##  6 Microsoft 2017-01-17 12:07:24 send
tag:finance.google.com,cluster:52779347599411
##  7 Microsoft 2017-01-17 12:07:24 msft
tag:finance.google.com,cluster:52779347599411
##  8 Microsoft 2017-01-17 12:07:24 stock
tag:finance.google.com,cluster:52779347599411
##  9 Microsoft 2017-01-17 12:07:24 soaring
tag:finance.google.com,cluster:52779347599411
## 10 Microsoft 2017-01-17 12:07:24 by
tag:finance.google.com,cluster:52779347599411
##    heading
##    <chr>
##  1 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  2 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  3 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  4 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  5 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  6 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  7 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  8 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
##  9 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
## 10 Microsoft Corporation: &quot;Data Privacy&quot; Could Send MSFT Stock
Soaring
## # ... with 105,044 more rows
```

Qui vediamo alcuni metadati di ciascun articolo accanto alle parole usate. Potremmo usare tf-idf per determinare quali parole erano più specifiche per ogni simbolo di azioni.

```
library(stringr)

stock_tf_idf <- stock_tokens %>%
  count(company, word) %>%
  filter(!str_detect(word, "\\d+")) %>%
  bind_tf_idf(word, company, n) %>%
  arrange(-tf_idf)
```

I termini principali per ciascuno sono visualizzati nella Figura 5.5. Come ci aspetteremmo, il nome e il simbolo della società sono in genere inclusi, ma lo sono anche molti dei loro prodotti e dirigenti, nonché le società con cui stanno facendo affari (come Disney con Netflix).

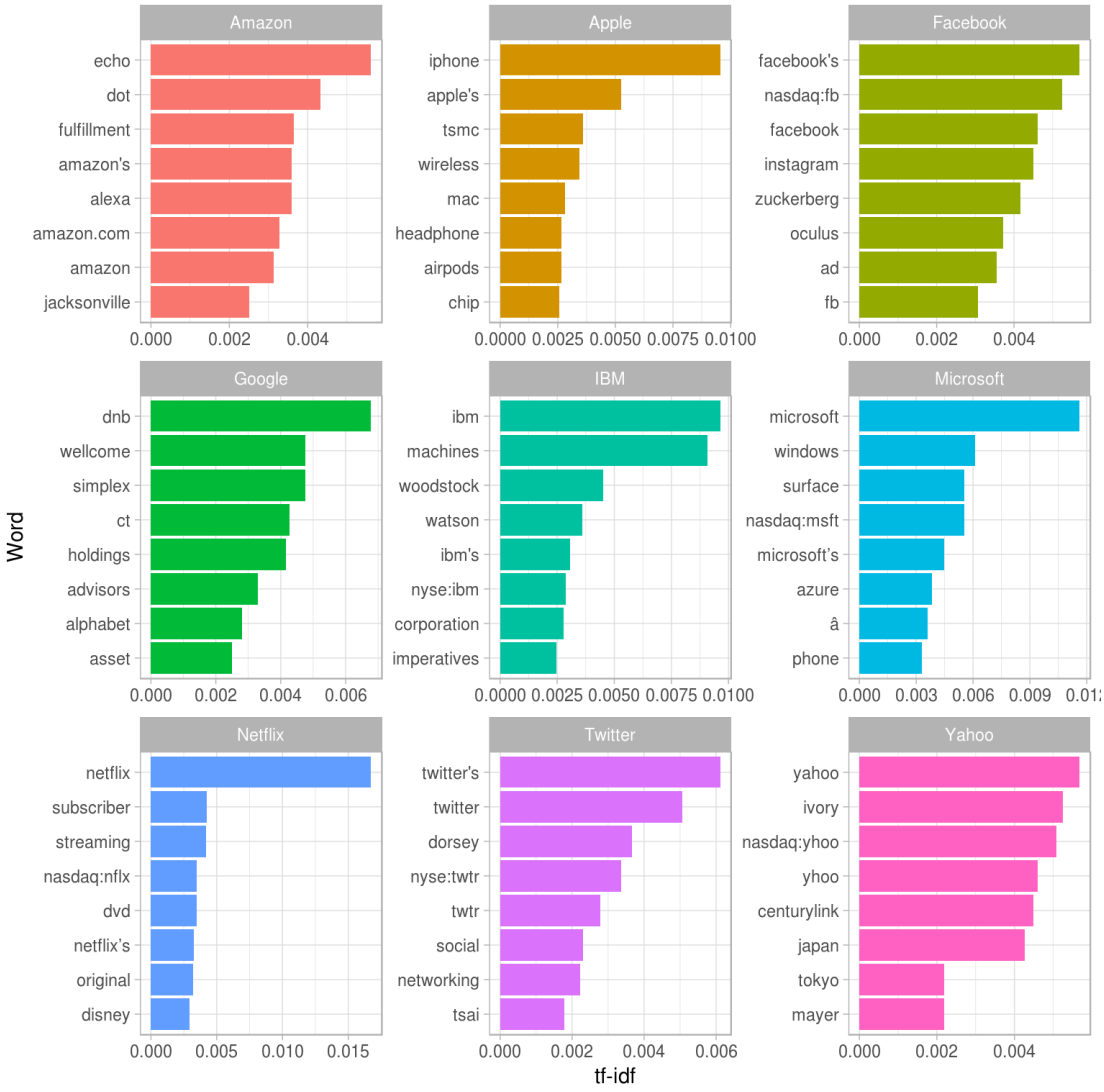


Figura 5.5: le 8 parole con il più alto tf-idf negli articoli recenti specifici per ciascuna azienda

Se fossimo interessati a utilizzare le notizie recenti per analizzare il mercato e prendere decisioni di investimento, probabilmente vorremmo utilizzare l'analisi del sentiment per determinare se la copertura delle notizie fosse positiva o negativa. Prima di eseguire tale analisi, dovremmo esaminare quali parole potrebbero contribuire maggiormente ai sentimenti positivi e negativi, come mostrato nel capitolo 2.4. Ad esempio, potremmo esaminarlo nel lessico AFINN (Figura 5.6).

```
stock_tokens %>%
  anti_join(stop_words, by = "word") %>%
  count(word, id, sort = TRUE) %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(word) %>%
  summarize(contribution = sum(n * score)) %>%
  top_n(12, abs(contribution)) %>%
  mutate(word = reorder(word, contribution)) %>%
  ggplot(aes(word, contribution)) +
  geom_col() +
  coord_flip() +
  labs(y = "Frequency of word * AFINN score")
```

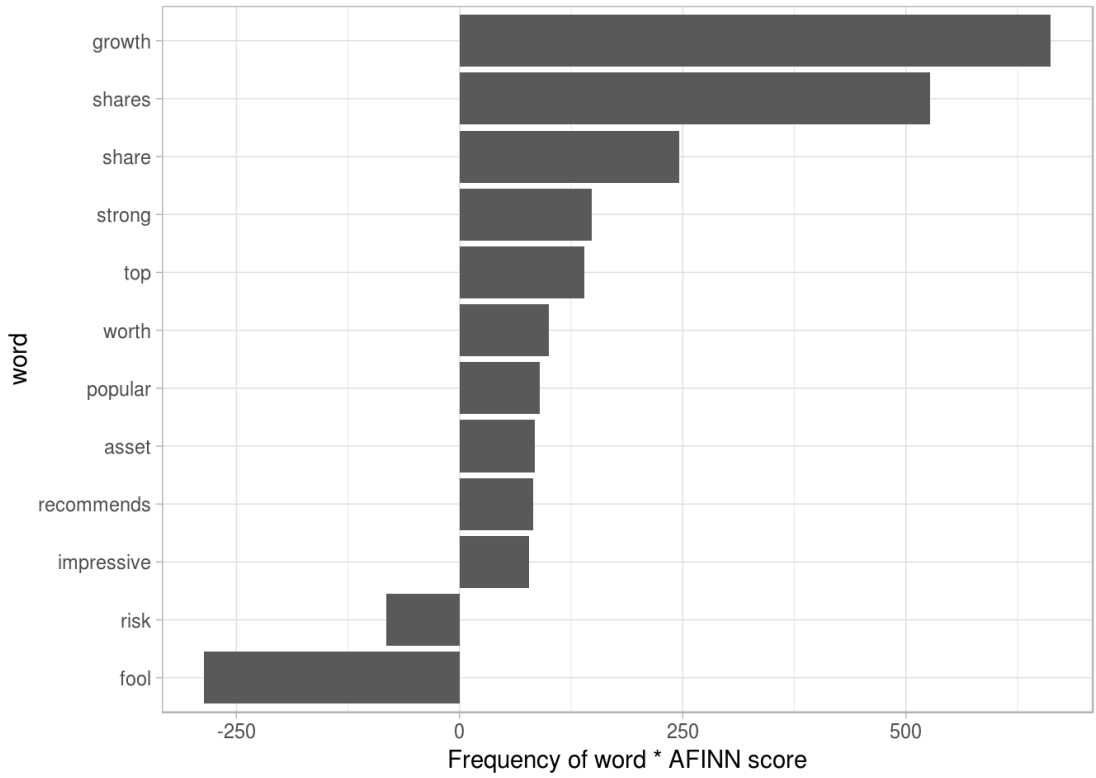


Figura 5.6: Le parole con il maggior contributo ai punteggi sentiment negli ultimi articoli finanziari, secondo il dizionario AFINN. Il "contributo" è il prodotto della parola e il punteggio di sentimento.

Nel contesto di questi articoli finanziari, ci sono alcune grandi bandiere rosse qui. Le parole "share" e "shares" sono contate come verbi positivi dal lessico AFINN ("Alice **condividerà** la sua torta con Bob"), ma in realtà sono nomi neutri ("Il prezzo delle azioni è di $ 12 per **azione** ") che potrebbero altrettanto facilmente essere in una frase positiva come negativa. La parola "pazzo" è ancora più ingannevole: si riferisce a Motley Fool, una società di servizi finanziari. In breve, possiamo vedere che il lessico del sentimento di AFINN è del tutto inadatto al contesto dei dati finanziari (come lo sono i lessici NRC e Bing).

Invece, introduciamo un altro lessico sul sentiment: il dizionario Loughran e McDonald dei termini di sentimento finanziario (Loughran e McDonald 2011) . Questo dizionario è stato sviluppato sulla base di analisi di relazioni finanziarie e evita intenzionalmente parole come "condivisione" e "sciocco", nonché termini più sottili come "responsabilità" e "rischio" che potrebbero non avere un significato negativo in un contesto finanziario.

I dati di Loughran dividono le parole in sei sentimenti: "positivo", "negativo", "litigioso", "incerto", "vincolante" e "superfluo". Potremmo iniziare esaminando le parole più comuni appartenenti a ciascun sentimento all'interno di questo set di dati di testo.

```
stock_tokens %>%
  count(word) %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  group_by(sentiment) %>%
  top_n(5, n) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
```

```
  geom_col() +
  coord_flip() +
  facet_wrap(~ sentiment, scales = "free") +
  ylab("Frequency of this word in the recent financial articles")
```

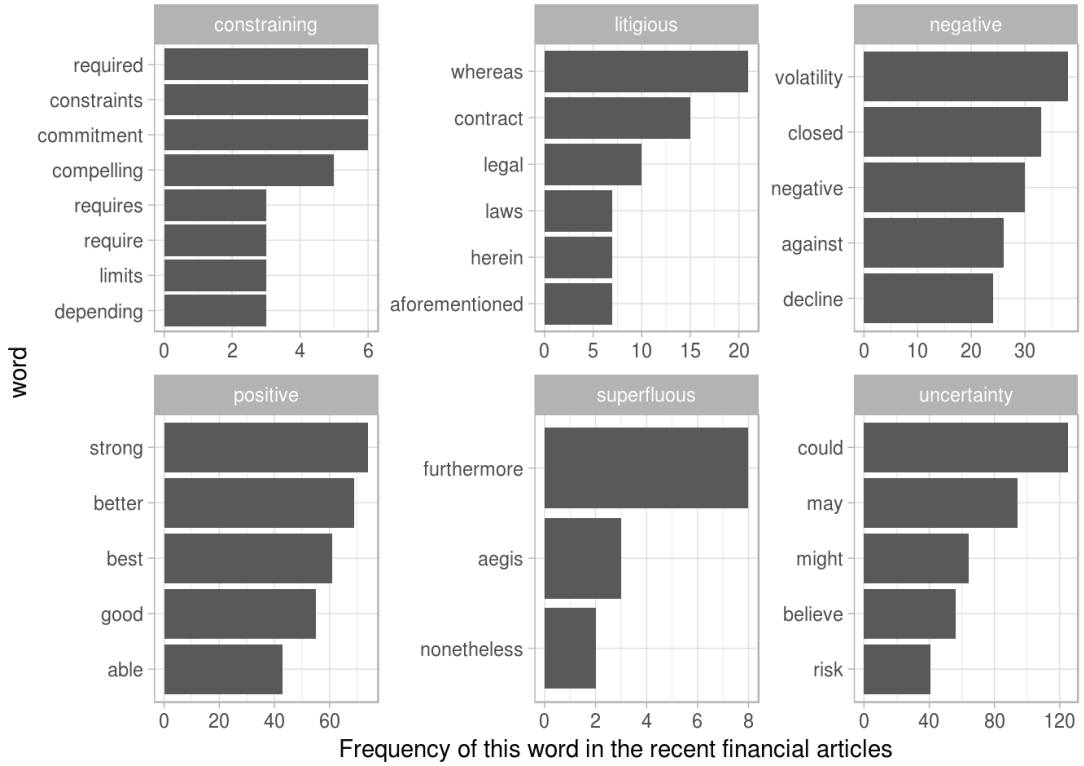


Figura 5.7: Le parole più comuni negli articoli di notizie finanziarie associati a ciascuno dei sei sentimenti nel lessico di Loughran e McDonald

Questi incarichi (figura 5.7) delle parole ai sentimenti sembrano più ragionevoli: le parole positive comuni includono "forte" e "migliore", ma non "condivisioni" o "crescita", mentre le parole negative includono "volatilità" ma non "sciocco". Anche gli altri sentimenti sembrano ragionevoli: i termini più comuni di "incertezza" includono "potrebbe" e "può".

Ora che sappiamo che possiamo fidarci del dizionario per approssimare i sentimenti degli articoli, possiamo usare i nostri metodi tipici per contare il numero di usi di ogni parola associata al sentimento in ogni corpus.

```
stock_sentiment_count <- stock_tokens %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  count(sentiment, company) %>%
  spread(sentiment, n, fill = 0)

stock_sentiment_count
```

```
## # A tibble: 9 x 7
##   company   constraining litigious negative positive superfluous uncertainty
##   <chr>            <dbl>     <dbl>    <dbl>    <dbl>       <dbl>       <dbl>
## 1 Amazon               7         8       84      144           3          70
## 2 Apple                9        11      161      156           2         132
## 3 Facebook             4        32      128      150           4          81
## 4 Google               7         8       60      103           0          58
## 5 IBM                  8        22      147      148           0         104
## 6 Microsoft            6        19       92      129           3         116
## 7 Netflix              4         7      111      162           0         106
```

```
## 8 Twitter              4        12       157        79           1          75
## 9 Yahoo                3        28       130        74           0          71
```

Potrebbe essere interessante esaminare quale compagnia abbia più notizie con termini "litigiosi" o "incerti". Ma la misura più semplice, come per la maggior parte delle analisi del capitolo 2, è vedere se le notizie sono più positive o negative. Come misura quantitativa generale del sentimento, useremo "(positivo - negativo) / (positivo + negativo)" (Figura 5.8).

```
stock_sentiment_count %>%
  mutate(score = (positive - negative) / (positive + negative)) %>%
  mutate(company = reorder(company, score)) %>%
  ggplot(aes(company, score, fill = score > 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(x = "Company",
       y = "Positivity score among 20 recent news articles")
```

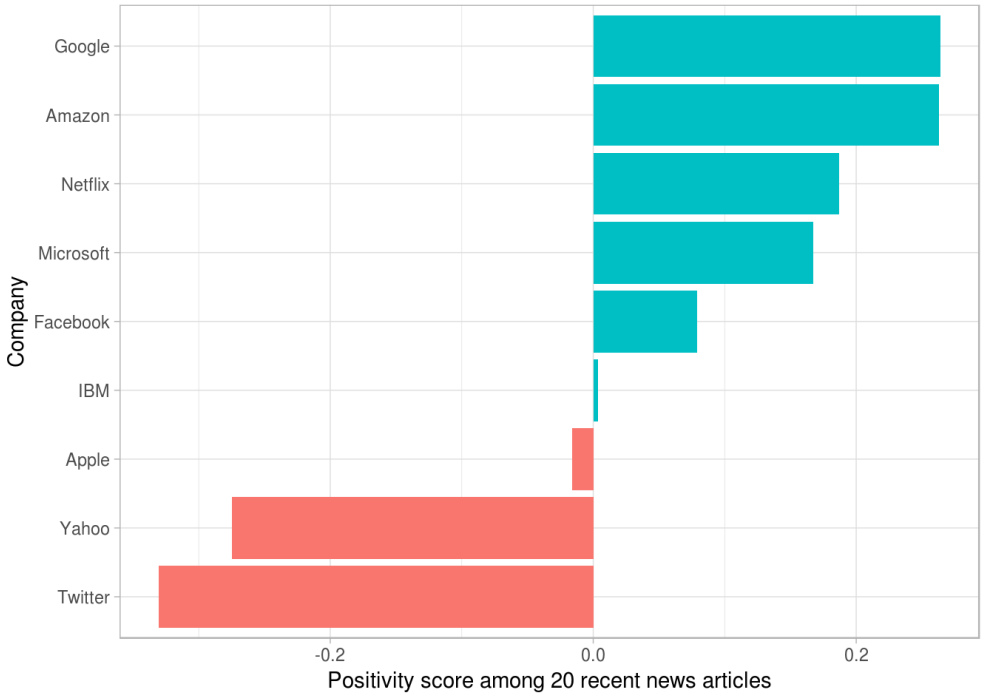


Figura 5.8: "Positività" della copertura di notizie intorno a ciascun titolo a gennaio 2017, calcolata come (positiva / negativa) / (positiva + negativa), in base all'utilizzo di parole positive e negative in 20 articoli di notizie recenti su ciascuna società.

Sulla base di questa analisi, diciamo che nel gennaio 2017 la maggior parte della copertura di Yahoo e Twitter è stata fortemente negativa, mentre la copertura di Google e Amazon è stata la più positiva. Uno sguardo agli attuali titoli finanziari suggerisce che è sulla strada giusta. Se fossi interessato ad ulteriori analisi, potresti utilizzare uno dei tanti pacchetti finanziari quantitativi di R per confrontare questi articoli con i prezzi delle azioni recenti e altre metriche.

5.4 Riepilogo

L'analisi del testo richiede di lavorare con una varietà di strumenti, molti dei quali hanno input e output che non sono in una forma ordinata. Questo capitolo ha mostrato come convertire tra un frame di dati di testo ordinato e matrici di termini documento sparse, nonché come riordinare un oggetto Corpus contenente metadati di documenti. Il prossimo capitolo mostrerà un altro esempio degno di nota di un pacchetto, topicmodels, che richiede come input una matrice di termini del documento, dimostrando che questi strumenti di conversione sono una parte essenziale dell'analisi del testo.

6 Modellazione degli argomenti (topic modelling)

TOPIC **MODELLING.** Partiamo da un corpus che contiene una collezione di documenti, vogliamo suddividere il corpus in un certo numero di topic (di cui non so l'esistenza a priori) in modo che i documenti che appartengono ad ognuno di questi topic siano simili tra di loro. È un po' il problema del clustering (cerco un certo numero di gruppi in modo che gli elementi all'interno del cluster siano il più possibili vicini tra di loro).

Nel text mining, spesso abbiamo raccolte di documenti, come post di blog o articoli di notizie, che vorremmo dividere in gruppi naturali in modo che possiamo comprenderli separatamente. La modellazione di argomenti è un metodo per la classificazione senza supervisione di tali documenti, simile al clustering su dati numerici, che trova gruppi naturali di elementi anche quando non siamo sicuri di ciò che stiamo cercando.

L'assegnazione latente di Dirichlet (**LDA**) è un metodo particolarmente popolare per il montaggio di un modello di argomento. Tratta ogni documento come una miscela di argomenti e ogni argomento come una miscela di parole. Ciò consente ai documenti di "sovrapporsi" in termini di contenuto, piuttosto che essere separati in gruppi discreti, in un modo che rispecchia l'uso tipico del linguaggio naturale.

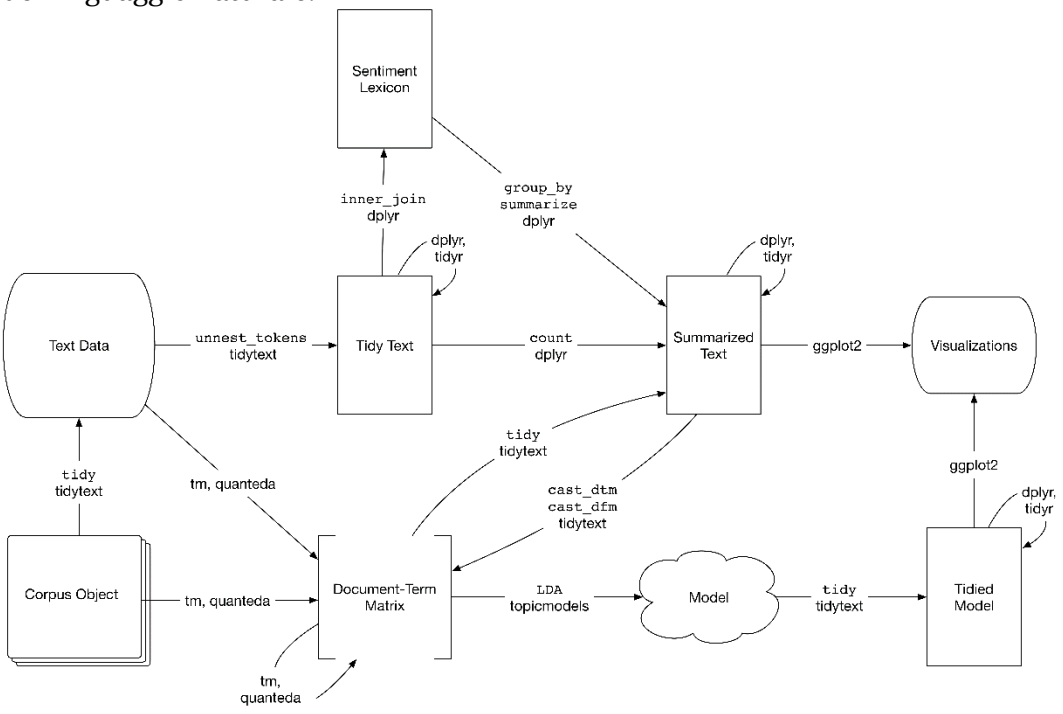


Figura 6.1: Un diagramma di flusso di un'analisi del testo che incorpora la modellazione dell'argomento. Il pacchetto topicmodels prende come input una Matrice del Documento-Termine e produce un modello che può essere tidato da tidytext, in modo tale che possa essere manipolato e visualizzato con dplyr e ggplot2.

Come mostrato nella Figura 6.1 , possiamo usare i principi del testo in ordine per approcciare la modellazione degli argomenti con lo stesso set di strumenti ordinati che abbiamo usato in questo libro. In questo capitolo impareremo a lavorare con gli oggetti LDA del pacchetto topicmodels , in particolare riordinando tali modelli in modo che possano essere manipolati con ggplot2 e dplyr. Esploreremo anche un esempio di capitoli di cluster provenienti da diversi libri, in cui possiamo vedere che un modello di argomento "impara" a distinguere i quattro libri in base al contenuto del testo.

6.1 Assegnazione di Dirichlet latente

Qui i punti però sono dei documenti, saranno simili se parlano di argomenti simili. Utilizzeremo la tecnica che si chiama **LDA** (Latent Dirichlet allocation); questa tecnica crea due suddivisioni. Ogni documento viene suddiviso in un certo numero di topic e ogni topic viene suddiviso in un numero di parole. Un documento appartiene con una certa probabilità ad un topic (80% al topic politica e 20% al topic finanza), non c'è una suddivisione esatta e non c'è una suddivisione esatta tra i topic e le parole (una parola potrebbe essere associata a più topic). Dobbiamo capire qual è il senso dei topic guardando le parole e capire quali sono i documenti che sono contenuti maggiormente in un certo topic. Le beta sono delle probabilità che sommano ad 1 all'interno del topic, ogni parola ha una Pr di appartenere a quel topic. Le gamma sono le Pr che un topic appartenga ad un documento e la somma delle gamma per ogni documento fa 1.

L'allocazione latente di Dirichlet è uno degli algoritmi più comuni per la modellazione degli argomenti. Senza immergerci nella matematica dietro al modello, possiamo capirlo come guidato da due principi.

- Ogni documento è un misto di argomenti. Immaginiamo che ogni documento possa contenere parole di diversi argomenti in proporzioni particolari. Ad esempio, in un modello a due argomenti potremmo dire "Il documento 1 è il 90% di argomento A e il 10% di argomento B, mentre il documento 2 è il 30% di argomento A e il 70% di argomento B."

- Ogni argomento è un misto di parole. Ad esempio, potremmo immaginare un modello a due argomenti di notizie americane, con un argomento per "politica" e uno per "intrattenimento". Le parole più comuni nel tema politico potrebbero essere "Presidente", "Congresso" e "governo" ", Mentre il tema dell'intrattenimento potrebbe essere costituito da parole come" film "," televisione "e" attore ". È importante sottolineare che le parole possono essere condivise tra argomenti; una parola come "budget" potrebbe apparire in entrambi allo stesso modo.

LDA è un metodo matematico per stimare entrambi allo stesso tempo: trovare la combinazione di parole che è associata a ciascun argomento, determinando allo stesso tempo la combinazione di argomenti che descrive ciascun documento. Esistono numerose implementazioni esistenti di questo algoritmo e ne esploreremo una in profondità.

Nel Capitolo 5 abbiamo brevemente introdotto il AssociatedPress set di dati fornito dal pacchetto topicmodels, come esempio di DocumentTermMatrix. Questa è una raccolta di 2246 articoli di notizie di un'agenzia di stampa americana, pubblicati per lo più intorno al 1988.

```
library(topicmodels)

data("AssociatedPress")
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Possiamo utilizzare la LDA() funzione dal pacchetto topicmodels, impostando k = 2, per creare un modello LDA a due argomenti.

Quasi tutti i modelli di argomento nella pratica useranno un più ampio k, ma vedremo presto che questo approccio di analisi si estende a un numero maggiore di argomenti.

Questa funzione restituisce un oggetto contenente i dettagli completi dell'adattamento del modello, ad esempio il modo in cui le parole sono associate agli argomenti e in che modo gli argomenti sono associati ai documenti.

```
# set a seed so that the output of the model is predictable
ap_lda <- LDA(AssociatedPress, k = 2, control = list(seed = 1234))
ap_lda
```

```
## A LDA_VEM topic model with 2 topics.
```

Adattare il modello è stata la "parte facile": il resto dell'analisi coinvolgerà l'esplorazione e l'interpretazione del modello utilizzando le funzioni di riordino del pacchetto tidytext.

6.1.1 Probabilità di argomento di parole

Nel Capitolo 5 abbiamo introdotto il tidy()metodo, originariamente dal pacchetto di scopa (Robinson 2017) , per riordinare gli oggetti del modello. Il pacchetto tidytext fornisce questo metodo per estrarre le probabilità per argomento per parola, chiamate β ("Beta"), dal modello.

```
library(tidytext)

ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
## # A tibble: 20,946 x 3
##    topic term           beta
##    <int> <chr>         <dbl>
##  1     1 aaron      1.69e-12
##  2     2 aaron      3.90e- 5
##  3     1 abandon    2.65e- 5
##  4     2 abandon    3.99e- 5
```

```
##  5     1 abandoned  1.39e- 4
##  6     2 abandoned  5.88e- 5
##  7     1 abandoning 2.45e-33
##  8     2 abandoning 2.34e- 5
##  9     1 abbott     2.13e- 6
## 10     2 abbott     2.97e- 5
## # ... with 20,936 more rows
```

Si noti che questo ha trasformato il modello in un formato a un argomento per riga per riga. Per ogni combinazione, il modello calcola la probabilità che quel termine sia generato da quell'argomento. Ad esempio, il termine "aaron" ha un 1.686917 × 10 - 121.686917×10−12probabilità di essere generata dall'argomento 1, ma 3.8959408 × 10 - 53.8959408×10−5 probabilità di essere generata dall'argomento 2.

Potremmo usare dplyr's top_n()per trovare i 10 termini più comuni all'interno di ogni argomento. Come una cornice dati ordinata, questo si presta bene a una visualizzazione ggplot2 (Figura 6.2).

```
library(ggplot2)
library(dplyr)

ap_top_terms <- ap_topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

ap_top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```

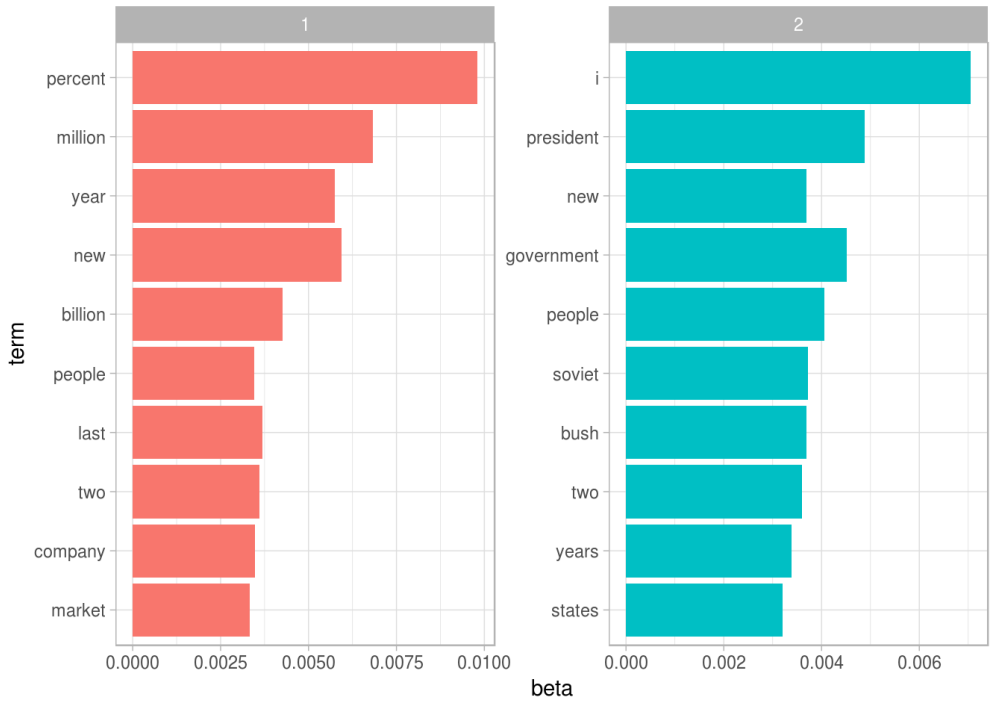


Figura 6.2: i termini più comuni all'interno di ciascun argomento

Questa visualizzazione ci consente di comprendere i due argomenti che sono stati estratti dagli articoli. Le parole più comuni nell'argomento 1 includono "percent", "million", "billion" e "company", che suggeriscono che potrebbe rappresentare notizie aziendali o finanziarie. Quelli più comuni nell'argomento 2 includono "presidente", "governo" e "sovietico", suggerendo che questo argomento rappresenta notizie politiche. Un'osservazione importante sulle parole di ciascun argomento è che alcune parole, come "nuovo" e "persone", sono comuni in entrambi gli argomenti. Questo è un vantaggio della modellazione dell'argomento rispetto ai metodi del "clustering duro": gli argomenti utilizzati nel linguaggio naturale potrebbero sovrapporsi in termini di parole.

In alternativa, potremmo considerare i termini che hanno avuto la *maggiore differenza* in βBtra argomento 1 e argomento 2. Questo può essere stimato in base al rapporto di registro dei due: $\log_2(\beta_2 \backslash \beta_1)$ (un rapporto di registro è utile perché fa la differenza simmetrica: β_2 essere due volte più grande porta ad un rapporto log di 1, mentre β_1 essere due volte più grandi risultati in -1). Per vincolarlo a un insieme di parole particolarmente rilevanti, possiamo filtrare per parole relativamente comuni, come quelle che hanno un βB maggiore di 1/1000 in almeno un argomento.

```
library(tidyr)

beta_spread <- ap_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2(topic2 / topic1))

beta_spread
```

```
## # A tibble: 198 x 4
##    term              topic1      topic2 log_ratio
##    <chr>              <dbl>       <dbl>     <dbl>
##  1 administration 0.000431  0.00138         1.68
##  2 ago            0.00107   0.000842       -0.339
##  3 agreement      0.000671  0.00104         0.630
##  4 aid            0.0000476 0.00105         4.46
##  5 air            0.00214   0.000297       -2.85
##  6 american       0.00203   0.00168        -0.270
##  7 analysts       0.00109   0.000000578   -10.9
##  8 area           0.00137   0.000231       -2.57
##  9 army           0.000262  0.00105         2.00
## 10 asked          0.000189  0.00156         3.05
## # ... with 188 more rows
```

Le parole con le maggiori differenze tra i due argomenti sono visualizzate nella Figura 6.3.

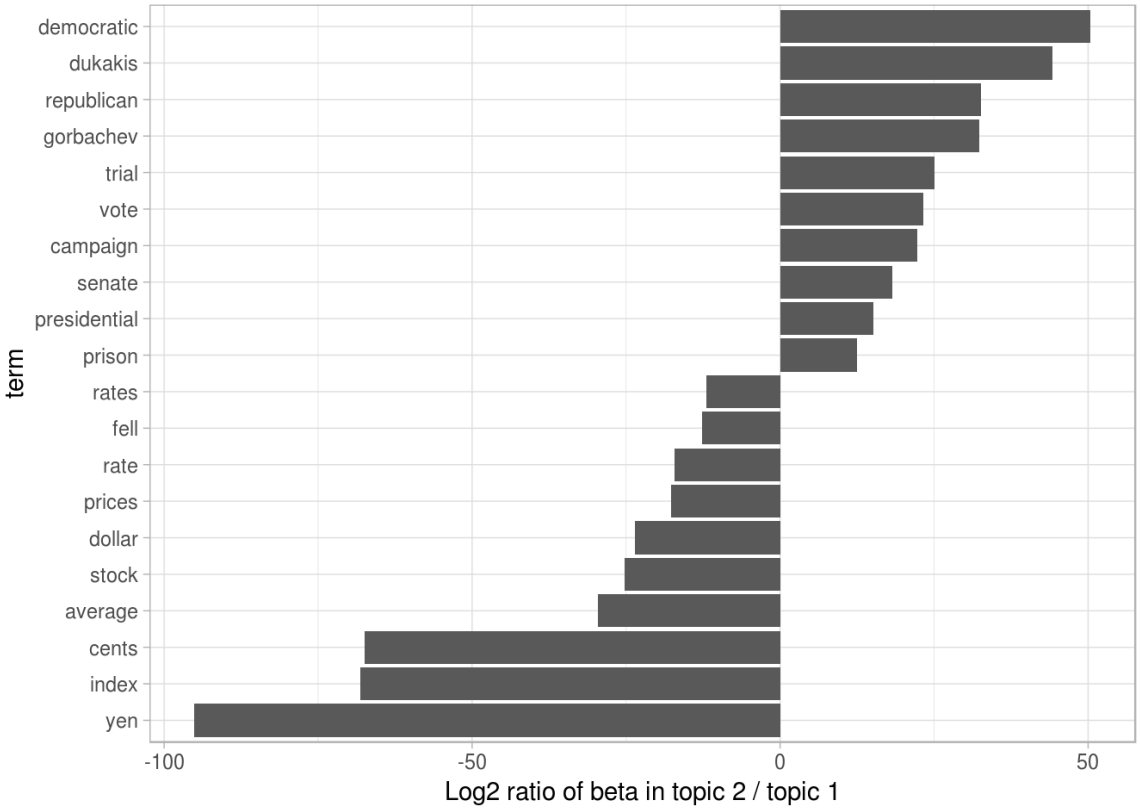


Figura 6.3: Parole con la maggiore differenza in β tra l'argomento 2 e l'argomento 1

Possiamo vedere che le parole più comuni nell'argomento 2 includono partiti politici come "democratico" e "repubblicano", così come nomi di politici come "dukakis" e "gorbaciov". L'argomento 1 era più caratterizzato da valute come "yen" e "dollaro", oltre a termini finanziari come "indice", "prezzi" e "tassi". Ciò aiuta a confermare che i due argomenti individuati dall'algoritmo erano notizie politiche e finanziarie.

6.1.2 Probabilità di argomento del documento

Oltre a stimare ogni argomento come una combinazione di parole, LDA modella anche ciascun documento come una combinazione di argomenti. Possiamo esaminare le probabilità per documento-per-argomento, chiamate γ ("Gamma"), con l' `matrix = "gamma"` argomento di tidy().

```
ap_documents <- tidy(ap_lda, matrix = "gamma")
ap_documents
## # A tibble: 4,492 x 3
##    document topic    gamma
##       <int> <int>    <dbl>
##  1        1     1 0.248
##  2        2     1 0.362
##  3        3     1 0.527
##  4        4     1 0.357
##  5        5     1 0.181
##  6        6     1 0.000588
##  7        7     1 0.773
##  8        8     1 0.00445
##  9        9     1 0.967
## 10       10     1 0.147
```

```
## # ... with 4,482 more rows
```

Ciascuno di questi valori è una percentuale stimata di parole di quel documento generate da quell'argomento. Ad esempio, il modello stima che solo il 24,8% delle parole nel documento 1 sia stato generato dall'argomento 1.

Possiamo vedere che molti di questi documenti sono stati tratti da un mix dei due argomenti, ma quel documento 6 è stato disegnato quasi interamente dall'argomento 2, con un γ dall'argomento 1 vicino a zero. Per verificare questa risposta, potremmo utilizzare `tidy()` la matrice dei termini del documento (vedere il Capitolo 5.1) e controllare quali erano le parole più comuni in quel documento.

```
tidy(AssociatedPress) %>%
  filter(document == 6) %>%
  arrange(desc(count))
## # A tibble: 287 x 3
##    document term           count
##       <int> <chr>          <dbl>
##  1        6 noriega           16
##  2        6 panama            12
##  3        6 jackson            6
##  4        6 powell             6
##  5        6 administration     5
##  6        6 economic           5
##  7        6 general            5
##  8        6 i                  5
##  9        6 panamanian         5
## 10        6 american           4
## # ... with 277 more rows
```

Basandosi sulle parole più comuni, questo sembra essere un articolo sulla relazione tra il governo americano e il dittatore panamense Manuel Noriega, il che significa che l'algoritmo aveva ragione nel collocarlo nell'argomento 2 (come notizie politico / nazionali).

6.2 Esempio: la grande rapina della biblioteca

Quando si esamina un metodo statistico, può essere utile provarlo in un caso molto semplice in cui si conosce la "risposta giusta". Ad esempio, potremmo raccogliere una serie di documenti che si riferiscono in modo definitivo a quattro argomenti separati, quindi eseguire la modellazione degli argomenti per vedere se l'algoritmo può correttamente distinguere i quattro gruppi. Questo ci permette di ricontrollare che il metodo è utile e di capire come e quando può andare storto. Proveremo questo con alcuni dati della letteratura classica.

Supponiamo che un vandalo si sia spezzato nel tuo studio e abbia distrutto quattro dei tuoi libri:

- Grandi speranze di Charles Dickens
- La guerra dei mondi di HG Wells
- Ventimila leghe sotto il mare di Jules Verne
- Orgoglio e pregiudizio di Jane Austen

Questo vandalo ha strappato i libri in singoli capitoli e li ha lasciati in una grande pila. Come possiamo ripristinare questi capitoli disorganizzati nei loro libri originali? Questo è un problema impegnativo poiché i singoli capitoli non sono etichettati: non sappiamo quali parole potrebbero distinguerli in gruppi. Useremo

quindi la modellazione dell'argomento per scoprire come i capitoli si raggruppano in argomenti distinti, ognuno dei quali (presumibilmente) rappresenta uno dei libri.

Ritireremo il testo di questi quattro libri utilizzando il pacchetto gutenbergr introdotto nel Capitolo 3.

```
titles <- c("Twenty Thousand Leagues under the Sea", "The War of the Worlds",
            "Pride and Prejudice", "Great Expectations")
library(gutenbergr)

books <- gutenberg_works(title %in% titles) %>%
  gutenberg_download(meta_fields = "title")
```

Come pre-elaborazione, li dividiamo in capitoli, usiamo tidytext `unnest_tokens()` per separarli in parole, quindi rimuoviamo `stop_words`. Trattiamo ogni capitolo come un "documento" separato, ognuno con un nome come Great `Expectations_1` o Pride and `Prejudice_11`. (In altre applicazioni, ogni documento potrebbe essere un articolo di giornale o un post di un blog).

```
library(stringr)

# divide into documents, each representing one chapter
by_chapter <- books %>%
  group_by(title) %>%
  mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case =
TRUE)))) %>%
  ungroup() %>%
  filter(chapter > 0) %>%
  unite(document, title, chapter)

# split into words
by_chapter_word <- by_chapter %>%
  unnest_tokens(word, text)

# find document-word counts
word_counts <- by_chapter_word %>%
  anti_join(stop_words) %>%
  count(document, word, sort = TRUE) %>%
  ungroup()

word_counts
```

```
## # A tibble: 104,721 x 3
##    document                 word        n
##    <chr>                    <chr>   <int>
##  1 Great Expectations_57    joe        88
##  2 Great Expectations_7     joe        70
##  3 Great Expectations_17    biddy      63
##  4 Great Expectations_27    joe        58
##  5 Great Expectations_38    estella    58
##  6 Great Expectations_2     joe        56
##  7 Great Expectations_23    pocket     53
##  8 Great Expectations_15    joe        50
##  9 Great Expectations_18    joe        50
## 10 The War of the Worlds_16 brother    50
```

```
## # ... with 104,711 more rows
```

6.2.1 LDA sui capitoli

In questo momento la nostra cornice dati `word_counts` è in ordine, con un termine per documento per fila, ma il pacchetto topicmodels richiede a DocumentTermMatrix. Come descritto nel Capitolo 5.2, possiamo lanciare una tabella da una token per riga in una `DocumentTermMatrix` con tidytext `cast_dtm()`.

```
chapters_dtm <- word_counts %>%
  cast_dtm(document, word, n)

chapters_dtm
## <<DocumentTermMatrix (documents: 193, terms: 18215)>>
## Non-/sparse entries: 104721/3410774
## Sparsity           : 97%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

Possiamo quindi utilizzare la funzione `LDA()` per creare un modello a quattro argomenti. In questo caso sappiamo che stiamo cercando quattro argomenti perché ci sono quattro libri; in altri problemi potremmo aver bisogno di provare alcuni valori diversi di `k`.

```
chapters_lda <- LDA(chapters_dtm, k = 4, control = list(seed = 1234))
chapters_lda

## A LDA_VEM topic model with 4 topics.
```

Come abbiamo fatto sui dati della Associated Press, possiamo esaminare le probabilità per argomento per parola.

```
chapter_topics <- tidy(chapters_lda, matrix = "beta")
chapter_topics

## # A tibble: 72,860 x 3
##    topic term        beta
##    <int> <chr>      <dbl>
##  1     1 joe     5.83e-17
##  2     2 joe     3.19e-57
##  3     3 joe     4.16e-24
##  4     4 joe     1.45e- 2
##  5     1 biddy   7.85e-27
##  6     2 biddy   4.67e-69
##  7     3 biddy   2.26e-46
##  8     4 biddy   4.77e- 3
##  9     1 estella 3.83e- 6
## 10     2 estella 5.32e-65
## # ... with 72,850 more rows
```

Si noti che questo ha trasformato il modello in un formato a un argomento per riga per riga. Per ogni combinazione, il modello calcola la probabilità che quel termine sia generato da quell'argomento. Ad

esempio, il termine "joe" ha una probabilità quasi zero di essere generato dagli argomenti 1, 2 o 3, ma costituisce l'1,45% dell'argomento 4.

Potremmo usare dplyr's `top_n()` per trovare i primi 5 termini all'interno di ogni argomento.

```
top_terms <- chapter_topics %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

top_terms
## # A tibble: 20 x 3
##    topic term         beta
##    <int> <chr>       <dbl>
##  1     1 elizabeth 0.0141
##  2     1 darcy     0.00881
##  3     1 miss      0.00871
##  4     1 bennet    0.00695
##  5     1 jane      0.00650
##  6     2 captain   0.0155
##  7     2 nautilus  0.0131
##  8     2 sea       0.00885
##  9     2 nemo      0.00871
## 10     2 ned       0.00803
## 11     3 people    0.00680
## 12     3 martians  0.00651
## 13     3 time      0.00535
## 14     3 black     0.00528
## 15     3 night     0.00448
## 16     4 joe       0.0145
## 17     4 time      0.00685
## 18     4 pip       0.00682
## 19     4 looked    0.00637
## 20     4 miss      0.00623
```

Questo output ordinato si presta bene a una visualizzazione ggplot2 (Figura 6.4).

```
library(ggplot2)

top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```

Figura 6.4: i termini più comuni all'interno di ciascun argomento

Questi argomenti sono chiaramente associati ai quattro libri! Non c'è dubbio che il tema di "capitano", "nautilus", "mare" e "nemo" appartiene a Ventimila leghe sotto i mari , e che "jane", "darcy" e "elisabetta" appartengono a Pride e Pregiudizio . Vediamo "pip" e "joe" da Great Expectations e "marziani", "nero" e "notte" da La guerra dei mondi. Notiamo inoltre che, in linea con l'LDA come metodo di "clustering fuzzy", possono esserci parole in comune tra più argomenti, come "miss" negli argomenti 1 e 4 e "time" negli argomenti 3 e 4.

6.2.2 Per-document classification

Ogni documento in questa analisi ha rappresentato un singolo capitolo. Pertanto, potremmo voler sapere quali argomenti sono associati a ciascun documento. Possiamo rimettere insieme i capitoli nei libri corretti? Possiamo trovarlo esaminando le probabilità per documento-per-argomento, γ ("gamma").

```
chapters_gamma <- tidy(chapters_lda, matrix = "gamma")
chapters_gamma

## # A tibble: 772 x 3
##    document                topic     gamma
##    <chr>                   <int>     <dbl>
##  1 Great Expectations_57       1 0.0000135
##  2 Great Expectations_7        1 0.0000147
##  3 Great Expectations_17       1 0.0000212
##  4 Great Expectations_27       1 0.0000192
```

```
##  5 Great Expectations_38        1 0.354
##  6 Great Expectations_2         1 0.0000172
##  7 Great Expectations_23        1 0.551
##  8 Great Expectations_15        1 0.0168
##  9 Great Expectations_18        1 0.0000127
## 10 The War of the Worlds_16     1 0.0000108
## # ... with 762 more rows
```

Ciascuno di questi valori è una percentuale stimata di parole di quel documento generate da quell'argomento. Ad esempio, il modello stima che ogni parola nel documento Great Expectations_57 abbia solo una probabilità dello 0,00135% di derivare dall'argomento 1 (Orgoglio e pregiudizio).

Ora che abbiamo queste probabilità di argomento, possiamo vedere quanto bene ha fatto il nostro apprendimento non supervisionato al momento di distinguere i quattro libri. Ci aspetteremmo che i capitoli all'interno di un libro risultino per lo più (o interamente), generati dall'argomento corrispondente.

Per prima cosa ri-separiamo il nome del documento in titolo e capitolo, dopo di che possiamo visualizzare la probabilità per documento per argomento per ciascuno (Figura 6.5).

```
chapters_gamma <- chapters_gamma %>%
  separate(document, c("title", "chapter"), sep = "_", convert = TRUE)

chapters_gamma
```

```
## # A tibble: 772 x 4
##    title                 chapter topic     gamma
##    <chr>                   <int> <int>     <dbl>
##  1 Great Expectations         57     1 0.0000135
##  2 Great Expectations          7     1 0.0000147
##  3 Great Expectations         17     1 0.0000212
##  4 Great Expectations         27     1 0.0000192
##  5 Great Expectations         38     1 0.354
##  6 Great Expectations          2     1 0.0000172
##  7 Great Expectations         23     1 0.551
##  8 Great Expectations         15     1 0.0168
##  9 Great Expectations         18     1 0.0000127
## 10 The War of the Worlds      16     1 0.0000108
## # ... with 762 more rows
```

```
# reorder titles in order of topic 1, topic 2, etc before plotting
chapters_gamma %>%
  mutate(title = reorder(title, gamma * topic)) %>%
  ggplot(aes(factor(topic), gamma)) +
  geom_boxplot() +
  facet_wrap(~ title)
```

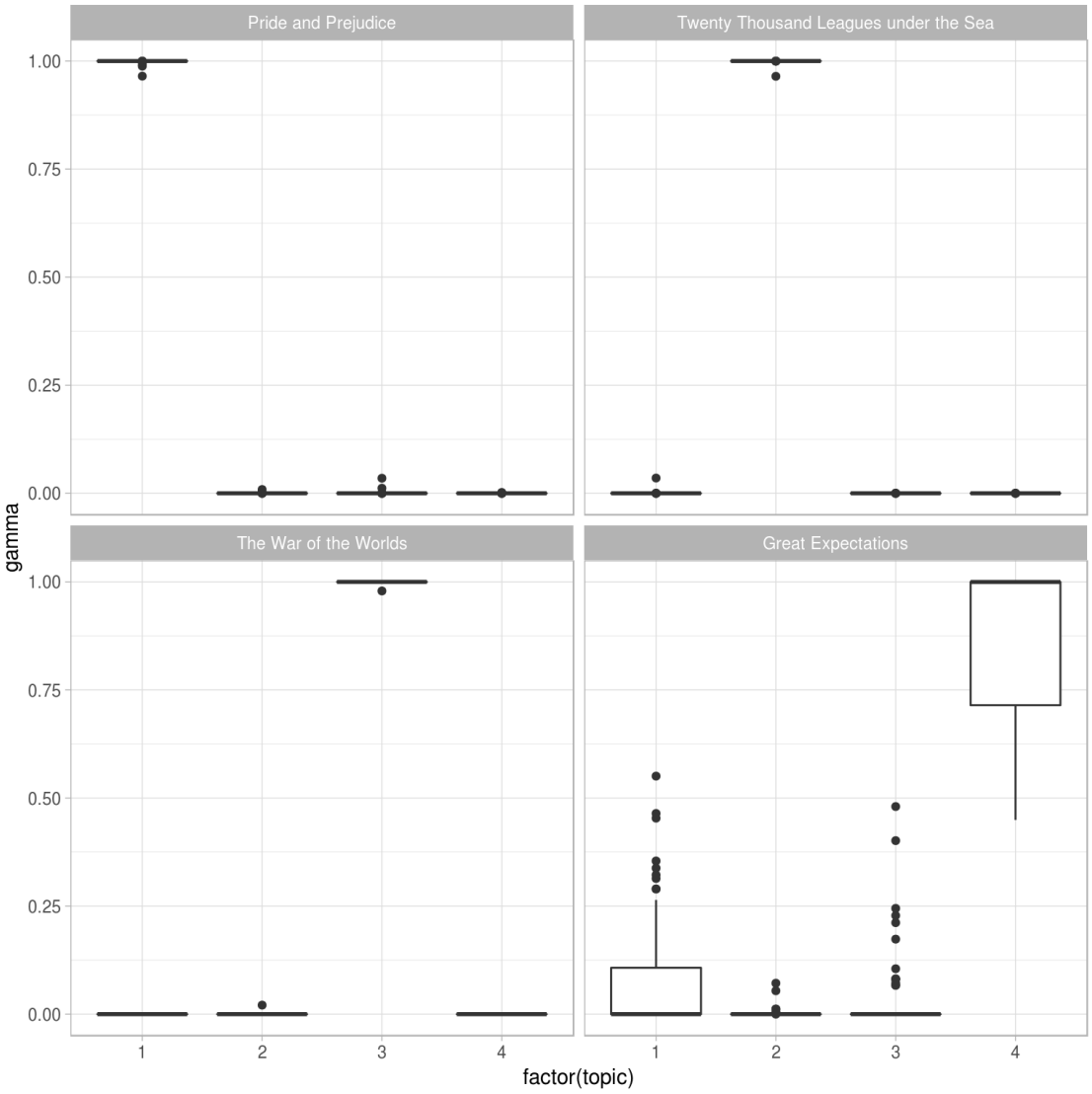


Figura 6.5: Le probabilità gamma per ogni capitolo all'interno di ogni libro

Notiamo che quasi tutti i capitoli di Pride and Prejudice, War of the Worlds e Twenty Thousand Leagues Under the Sea sono stati identificati in modo univoco come un singolo argomento ciascuno.

Sembra che alcuni capitoli di Great Expectations (che dovrebbero essere l'argomento 4) fossero in qualche modo associati ad altri argomenti. Ci sono casi in cui l'argomento più associato a un capitolo apparteneva a un altro libro? Per prima cosa troveremmo l'argomento più associato a ciascun capitolo usando `top_n()`, che è effettivamente la "classificazione" di quel capitolo.

```
chapter_classifications <- chapters_gamma %>%
  group_by(title, chapter) %>%
  top_n(1, gamma) %>%
  ungroup()

chapter_classifications
```

```
## # A tibble: 193 x 4
##    title                chapter topic gamma
##    <chr>                  <int> <int> <dbl>
```

```
##  1 Great Expectations       23     1 0.551
##  2 Pride and Prejudice      43     1 1.000
##  3 Pride and Prejudice      18     1 1.000
##  4 Pride and Prejudice      45     1 1.000
##  5 Pride and Prejudice      16     1 1.000
##  6 Pride and Prejudice      29     1 1.000
##  7 Pride and Prejudice      10     1 1.000
##  8 Pride and Prejudice       8     1 1.000
##  9 Pride and Prejudice      56     1 1.000
## 10 Pride and Prejudice      47     1 1.000
## # ... with 183 more rows
```

Possiamo quindi confrontare ciascun argomento di "consenso" per ciascun libro (l'argomento più comune tra i suoi capitoli) e vedere quali sono stati più spesso identificati erroneamente.

```
book_topics <- chapter_classifications %>%
  count(title, topic) %>%
  group_by(title) %>%
  top_n(1, n) %>%
  ungroup() %>%
  transmute(consensus = title, topic)

chapter_classifications %>%
  inner_join(book_topics, by = "topic") %>%
  filter(title != consensus)
```

```
## # A tibble: 2 x 5
##   title              chapter topic gamma consensus
##   <chr>                <int> <int> <dbl> <chr>
## 1 Great Expectations      23     1 0.551 Pride and Prejudice
## 2 Great Expectations      54     3 0.480 The War of the Worlds
```

Vediamo che solo due capitoli da Great Expectations sono stati classificati in modo errato, poiché la LDA ne ha descritto uno come proveniente dall'argomento "Orgoglio e pregiudizio" (argomento 1) e uno da La guerra dei mondi (argomento 3). Non è male per il clustering senza supervisione!

6.2.3 Assegnazione di parole: `augment`

Un passo dell'algoritmo LDA sta assegnando ogni parola in ciascun documento a un argomento. Più parole in un documento vengono assegnate a quell'argomento, generalmente, più peso (`gamma`) andrà su quella classificazione argomento-documento.

Potremmo voler prendere le coppie di documenti-parole originali e trovare quali parole in ciascun documento sono state assegnate a quale argomento. Questo è il lavoro della funzione `augment()`, che ha avuto origine anche nel pacchetto broom come metodo per riordinare l'output del modello. Mentre `tidy()` recupera i componenti statistici del modello, utilizza `augment()` un modello per aggiungere informazioni a ciascuna osservazione nei dati originali.

```
assignments <- augment(chapters_lda, data = chapters_dtm)
assignments
```

```
## # A tibble: 104,721 x 4
##    document              term  count .topic
##    <chr>                 <chr> <dbl>  <dbl>
##  1 Great Expectations_57 joe      88      4
##  2 Great Expectations_7  joe      70      4
##  3 Great Expectations_17 joe       5      4
##  4 Great Expectations_27 joe      58      4
##  5 Great Expectations_2  joe      56      4
##  6 Great Expectations_23 joe       1      4
##  7 Great Expectations_15 joe      50      4
##  8 Great Expectations_18 joe      50      4
##  9 Great Expectations_9  joe      44      4
## 10 Great Expectations_13 joe      40      4
## # ... with 104,711 more rows
```

Ciò restituisce una cornice di dati ordinata dei conteggi termine dei libri, ma aggiunge una colonna in più: `.topic` con l'argomento è stato assegnato ogni termine all'interno di ciascun documento. (Colonne aggiuntive aggiunte `augment` dall'inizio sempre `.` , per evitare di sovrascrivere le colonne esistenti). Possiamo combinare questa assignmentstabella con i titoli del libro di consenso per trovare quali parole sono state classificate erroneamente.

```
assignments <- assignments %>%
  separate(document, c("title", "chapter"), sep = "_", convert = TRUE) %>%
  inner_join(book_topics, by = c(".topic" = "topic"))

assignments
```

```
## # A tibble: 104,721 x 6
##    title              chapter term  count .topic consensus
##    <chr>                <int> <chr> <dbl>  <dbl> <chr>
##  1 Great Expectations      57 joe      88      4 Great Expectations
##  2 Great Expectations       7 joe      70      4 Great Expectations
##  3 Great Expectations      17 joe       5      4 Great Expectations
##  4 Great Expectations      27 joe      58      4 Great Expectations
##  5 Great Expectations       2 joe      56      4 Great Expectations
##  6 Great Expectations      23 joe       1      4 Great Expectations
##  7 Great Expectations      15 joe      50      4 Great Expectations
##  8 Great Expectations      18 joe      50      4 Great Expectations
##  9 Great Expectations       9 joe      44      4 Great Expectations
## 10 Great Expectations      13 joe      40      4 Great Expectations
## # ... with 104,711 more rows
```

Questa combinazione del vero libro (`title`) e del libro ad essa assegnato (`consensus`) è utile per ulteriori esplorazioni. Possiamo, ad esempio, visualizzare una **matrice di confusione**, mostrando quanto spesso le parole di un libro sono state assegnate a un altro, usando dplyr's `count()` e ggplot2 `geom_tile` (Figura 6.6) .

```
assignments %>%
  count(title, consensus, wt = count) %>%
  group_by(title) %>%
  mutate(percent = n / sum(n)) %>%
  ggplot(aes(consensus, title, fill = percent)) +
```

```
  geom_tile() +
  scale_fill_gradient2(high = "red", label = percent_format()) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        panel.grid = element_blank()) +
  labs(x = "Book words were assigned to",
       y = "Book words came from",
       fill = "% of assignments")
```

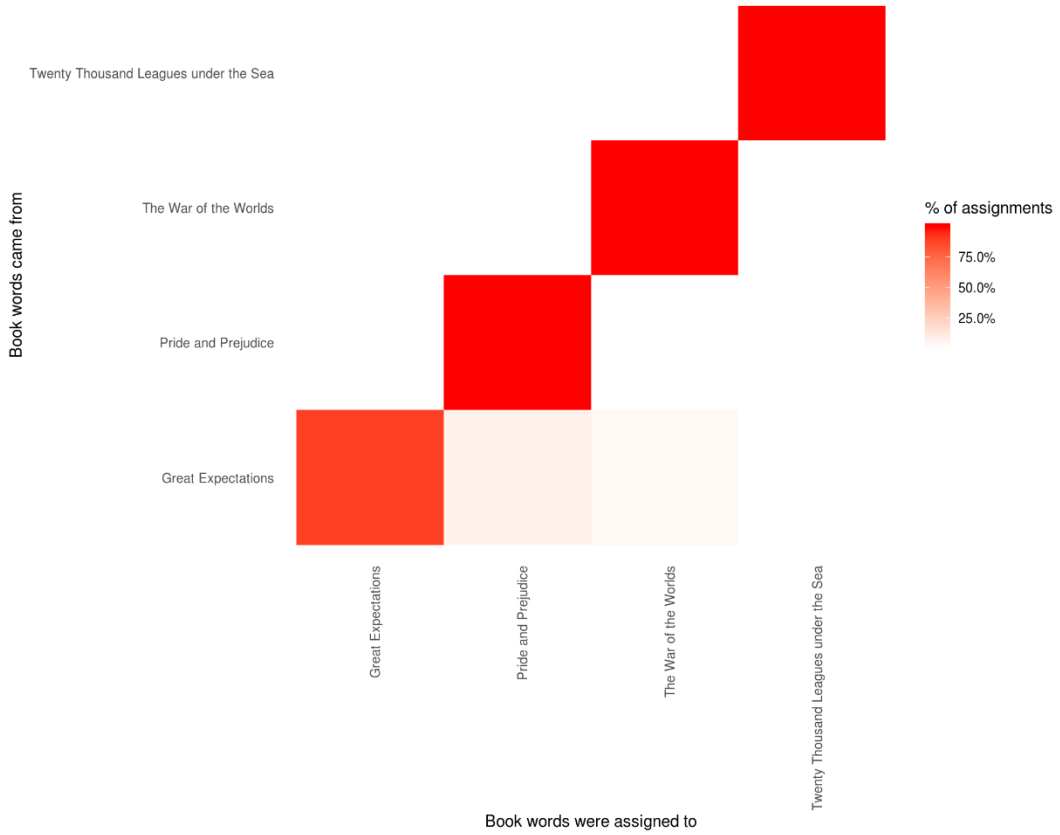


Figura 6.6: Matrice di confusione che mostra dove LDA ha assegnato le parole da ciascun libro. Ogni riga di questa tabella rappresenta il vero libro da cui proviene ogni parola, e ogni colonna rappresenta a quale libro è stato assegnato.

Notiamo che quasi tutte le parole di Pride and Prejudice, Twenty Thousand Leagues Under the Sea e War of the Worlds sono state assegnate correttamente, mentre Great Expectations ha avuto un buon numero di parole errate (che, come abbiamo visto sopra, hanno portato a due capitoli ottenere errori di classificazione).

Quali erano le parole più comunemente sbagliate?

```
wrong_words <- assignments %>%
  filter(title != consensus)

wrong_words

## # A tibble: 4,535 x 6
##    title                                chapter term     count .topic
##    <chr>                                  <int> <chr>    <dbl>  <dbl>
```

```
##  1 Great Expectations                           38 brother      2      1
##  2 Great Expectations                           22 brother      4      1
##  3 Great Expectations                           23 miss         2      1
##  4 Great Expectations                           22 miss        23      1
##  5 Twenty Thousand Leagues under the Sea         8 miss         1      1
##  6 Great Expectations                           31 miss         1      1
##  7 Great Expectations                            5 sergeant    37      1
##  8 Great Expectations                           46 captain      1      2
##  9 Great Expectations                           32 captain      1      2
## 10 The War of the Worlds                        17 captain      5      2
##    consensus
##    <chr>
##  1 Pride and Prejudice
##  2 Pride and Prejudice
##  3 Pride and Prejudice
##  4 Pride and Prejudice
##  5 Pride and Prejudice
##  6 Pride and Prejudice
##  7 Pride and Prejudice
##  8 Twenty Thousand Leagues under the Sea
##  9 Twenty Thousand Leagues under the Sea
## 10 Twenty Thousand Leagues under the Sea
## # ... with 4,525 more rows
```

```
wrong_words %>%
  count(title, consensus, term, wt = count) %>%
  ungroup() %>%
  arrange(desc(n))
```

```
## # A tibble: 3,500 x 4
##    title              consensus             term         n
##    <chr>              <chr>                 <chr>    <dbl>
##  1 Great Expectations Pride and Prejudice   love        44
##  2 Great Expectations Pride and Prejudice   sergeant    37
##  3 Great Expectations Pride and Prejudice   lady        32
##  4 Great Expectations Pride and Prejudice   miss        26
##  5 Great Expectations The War of the Worlds boat        25
##  6 Great Expectations Pride and Prejudice   father      19
##  7 Great Expectations The War of the Worlds water       19
##  8 Great Expectations Pride and Prejudice   baby        18
##  9 Great Expectations Pride and Prejudice   flopson     18
## 10 Great Expectations Pride and Prejudice   family      16
## # ... with 3,490 more rows
```

Possiamo vedere che un certo numero di parole sono state spesso assegnate al cluster Pride and Prejudice o War of the Worlds anche quando sono apparse in Great Expectations. Per alcune di queste parole, come "amore" e "signora", è perché sono più comuni in Orgoglio e pregiudizio (potremmo confermarlo esaminando i conteggi).

D'altra parte, ci sono alcune parole erroneamente classificate che non sono mai apparse nel romanzo per cui sono state erroneamente assegnate. Ad esempio, possiamo confermare che "flopson" appare solo in Great Expectations , anche se è assegnato al cluster "Pride and Prejudice".

```
word_counts %>%
  filter(word == "flopson")

## # A tibble: 3 x 3
##   document              word        n
##   <chr>                 <chr>   <int>
## 1 Great Expectations_22 flopson    10
## 2 Great Expectations_23 flopson     7
## 3 Great Expectations_33 flopson     1
```

L'algoritmo LDA è stocastico e può accidentalmente cadere su un argomento che si estende su più libri.

6.3 Implementazioni alternative LDA

La LDA()funzione nel pacchetto topicmodels è solo un'implementazione dell'algoritmo di allocazione di Dirichlet latente. Ad esempio, il pacchetto mallet (Mimno 2013) implementa un wrapper attorno al pacchetto Java MALLET per gli strumenti di classificazione del testo e il pacchetto tidytext fornisce anche tidier per questo output del modello.

Il pacchetto mallet assume un approccio alquanto diverso rispetto al formato di input. Ad esempio, prende i documenti non tokenizzati ed esegue la tokenizzazione stessa e richiede un file separato di stopword. Ciò significa che dobbiamo comprimere il testo in una stringa per ogni documento prima di eseguire l'LDA.

```
library(mallet)

# create a vector with one string per chapter
collapsed <- by_chapter_word %>%
  anti_join(stop_words, by = "word") %>%
  mutate(word = str_replace(word, "'", "")) %>%
  group_by(document) %>%
  summarize(text = paste(word, collapse = " "))

# create an empty file of "stopwords"
file.create(empty_file <- tempfile())
docs <- mallet.import(collapsed$document, collapsed$text, empty_file)

mallet_model <- MalletLDA(num.topics = 4)
mallet_model$loadDocuments(docs)
mallet_model$train(100)
```

Una volta creato il modello, tuttavia, possiamo utilizzare le funzioni `tidy()` e `augment()` descritte nel resto del capitolo in un modo quasi identico. Ciò include l'estrazione delle probabilità delle parole all'interno di ciascun argomento o argomento all'interno di ciascun documento.

```
# word-topic pairs
tidy(mallet_model)

# document-topic pairs
tidy(mallet_model, matrix = "gamma")
```

```
# column needs to be named "term" for "augment"
term_counts <- rename(word_counts, term = word)
augment(mallet_model, term_counts)
```

Potremmo usare ggplot2 per esplorare e visualizzare il modello nello stesso modo in cui abbiamo fatto l'output LDA.

6.4 Riepilogo

Questo capitolo introduce la modellazione di argomenti per la ricerca di gruppi di parole che caratterizzano un insieme di documenti e mostra come il tidy()verbo ci consente di esplorare e comprendere questi modelli usando dplyr e ggplot2. Questo è uno dei vantaggi dell'approccio ordinato all'esplorazione del modello: le sfide dei diversi formati di output sono gestite dalle funzioni di riordino e possiamo esplorare i risultati del modello utilizzando un set standard di strumenti. In particolare, abbiamo visto che la modellazione dell'argomento è in grado di separare e distinguere i capitoli da quattro libri separati, esplorando le limitazioni del modello trovando parole e capitoli assegnati in modo errato.

Esercitazione Mining financial articles.

Cercare di capire, in base al sentimento degli articoli, quali titoli sono positivi e quali sono negativi. Utilizzeremo un pacchetto tm.plugin.webmining che scarica alcuni articoli. Usa il lessico get_sentiments("loughran") che è specializzato per la finanza.