Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in

Statistica per le Tecnologie e le Scienze

RELAZIONE FINALE

UN'ANALISI DEI DATI DA TWITTER

Relatore Prof. Massimo Melucci
Dipartimento di Ingegneria dell'Informazione

Laureando: Elisa De Muri
Matricola N. 1190182

Anno Accademico 2021/2022

Indice

Introduzione

Nei mesi di ottobre e novembre del 2021 ho svolto uno stage presso **Omicron Consulting S.r.l.**, azienda che si occupa di *Information* e *Communication Technology*.

Le motivazioni che mi hanno spinta ad intraprendere un'esperienza lavorativa in questa fase del mio percorso di studi sono state la volontà di mettere in pratica le conoscenze teoriche acquisite durante gli anni di Università e quella di mettermi in gioco in un contesto per me nuovo come quello aziendale.

Il compito che mi è stato assegnato durante lo stage ha riguardato la costruzione di una procedura standardizzata che andasse a valutare l'opinione generale degli utenti di diversi social network riguardo un *product* di riferimento. L'obiettivo ultimo del progetto era quello di effettuare una *Sentiment Analysis* sui dati, andando a classificare i commenti degli utenti secondo diverse fasce di gradimento.
Oltre ad ottenere una stima dell'opinione degli utenti riguardo al *product* preso in considerazione, la procedura ha permesso di differenziare i risultati in base ai diversi aspetti del prodotto e, di conseguenza, di contestualizzare i giudizi positivi o negativi.

La procedura è stata implementata con i dati relativi ad *iPhone 13*, l'ultimo modello di smartphone prodotto da Apple e messo sul mercato nel settembre del 2021. È stato scelto questo come prodotto esemplificativo proprio perché, a inizio analisi, rappresentava un argomento di tendenza, con la possibilità di raccogliere molte informazioni.
In questo caso specifico il *product* scelto come riferimento per l'analisi è un bene materiale offerto e messo in commercio da un'azienda, ma le stesse tecniche presentate possono essere utilizzate per qualsiasi argomento che

generi una discussione tra gli utenti, come ad esempio un servizio, un personaggio pubblico, un tema sociale, politico o sportivo.

Il progetto iniziale ambiva ad esplorare diversi social media, ma a causa delle tempistiche dello stage, è stato possibile prendere in considerazione solo i dati provenienti da Twitter e Reddit. Questi due social network, infatti, non hanno richiesto particolari autorizzazioni per poter accedere alle loro *API* ed elaborare i dati.

Twitter è un social network che offre un servizio di notizie e *microblogging*, sul quale gli utenti postano e interagiscono con brevi messaggi di testo lunghi al massimo 280 caratteri, chiamati *tweet*. I tweet possono contenere file multimediali, tra cui foto, video e messaggi audio. A gennaio del 2022, Twitter conta 396.5 milioni di utenti, di cui 206 milioni sono utenti attivi giornalmente. [1]

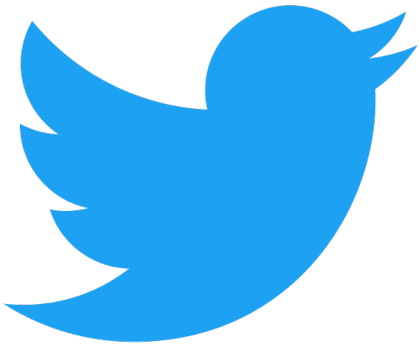


Figura 1: Logo di Twitter

Reddit è un sito Internet strutturato come un forum, dove gli utenti pubblicano contenuti sotto forma di post. Anche su Reddit, come su Twitter, si possono trovare molti generi di contenuti, dall'attualità all'intrattenimento, che vengono poi raggruppati per aree di interesse nei cosiddetti *subreddit*. I dati relativi al numero di utenti attivi su Reddit non sono aggiornati, per decisione del social stesso, ma ad ottobre del 2020 erano 52 milioni, con una crescita del 44% rispetto all'anno precedente. [2]

Figura 2: Logo di Reddit

[1] Dean B., Backlinko, *How Many People Use Twitter in 2022? [New Twitter Stats]*, `https://backlinko.com/twitter-users`

[2] Stentella U., Lega Nerd, *Reddit ha svelato per la prima volta quanti utenti ha*, `https://leganerd.com/2020/12/04/`

La procedura sviluppata, dopo aver acquisito, filtrato ed elaborato i dati grezzi dai social, ha previsto la normalizzazione di questi e il loro inserimento in una tabella unica. Composto dunque un solido dataframe, sono state effettuate le analisi che hanno incluso la costruzione di cluster e l'individuazione di regole di associazione. All'interno dei cluster, ogni commento è stato classificato secondo il giudizio espresso dall'utente.

Durante l'attività sono state utilizzate tecniche di *Machine Learning* e di *Data Mining*, in quanto vi era necessità di elaborare una notevole quantità di dati.

Data Mining non ha una vera e propria traduzione in italiano, ma per *mining* si intende l'estrazione di materiale prezioso dalle miniere. Si può dunque pensare al *Data Mining* come ad una "estrazione di dati". Comprende una serie di tecniche e metodi non supervisionati per la rilevazione di *pattern*, ovvero dati complessi, che arricchiscono l'informazione disponibile permettendo di scoprire informazioni non note a priori.
Il *Data Mining* è stato utilizzato per la ricerca di regole di associazione e per il *clustering*.

Il **Machine Learning** si differenzia dal *Data Mining* perché comprende tecniche supervisionate di elaborazione dei dati. La realtà di interesse è vista come la relazione tra n punti di uno spazio reale a più dimensioni: un metodo di Machine Learning assegna un punto x dello spazio $\mathbb{R}^k$ ad un punto y dello spazio $\mathbb{R}^h$, dove x è una caratteristica (o *feature*) e y è il *pattern*. Alla macchina viene dato, a priori, un *training set* come input, del tipo $\{(x1, y1), ..., (xn, yn)\}$. Grazie a queste istruzioni, è poi in grado di stimare e prevedere delle relazioni tra le variabili. [3]
Una delle tecniche di *Machine Learning* più comune è la classificazione, che è stata usata in diversi momenti dell'attività. Per esempio, i dati raccolti dai social media sono stati classificati in modo tale che fossero più pertinenti possibile con l'argomento di nostro effettivo interesse; un'altra classificazione è stata svolta nelle fasi finali della procedura, con la *Sentiment Analysis*, dove i tweet e i post sono stati classificati come positivi, negativi o neutri. Si può dunque affermare che il *Machine Learning*

`reddit-ha-svelato-per-la-prima-volta-quanti-utenti-ha/`

[3] Melucci M. *Lezioni di Basi di Dati*. Padova: C.L.E.U.P., 2020

sia stato utilizzato nel corso di tutta l'attività di stage, rendendolo un elemento chiave del progetto.

La trattazione seguente è divisa in due parti principali:

- **Capitolo 1**: verte sulla descrizione dell'esperienza di stage, con una panoramica sull'azienda in cui ho lavorato, le modalità di svolgimento del lavoro e gli strumenti e i programmi utilizzati;
- **Capitolo 2**: è focalizzato sulle attività svolte, cioè contiene la presentazione della procedura implementata e i risultati trovati con essa.

Capitolo 1

L'esperienza di stage

Per avere un'idea più chiara dell'esperienza di stage che ho intrapreso, vengono presentati gli aspetti principali, a partire da una breve presentazione dell'azienda *Omicron Consulting* e di ciò di cui si occupa, all'ambiente e le modalità in cui ho svolto il lavoro e gli strumenti utilizzati al fine della buona riuscita del progetto.

1.1 Contesto aziendale

Figura 1.1: Logo di *Omicron Consulting*

Omicron Consulting è un'azienda che, da oltre 40 anni, si occupa di *Information e Communication Technology*. Conta circa 300 dipendenti distruibuiti nel territorio nazionale in diverse sedi: a quella di Torino, prima e storica sede, che ancora oggi rappresenta la sede legale dove risiede la direzione generale, sono state affiancate le sedi di Milano, Padova, Roma, Napoli e Brindisi. Negli ultimi anni l'azienda si è espansa anche in Europa, aprendo ulteriori sedi inizialmente a Lugano (Svizzera), successivamente a Madrid e a Londra.

Omicron Consulting si pone come *System Integrator* per aziende di medie e grandi dimensioni, cioè fornitore di prodotti, sistemi, tecnologie e soluzioni integrate, per accompagnarle verso una digitalizzazione che possa aumentare la loro produttività ed efficienza. I settori di mercato di cui si occupa sono principalmente quello bancario, assicurativo e industriale, comprendendo anche gli ambiti manifatturiero, delle telecomunicazioni, dei servizi, del turismo. [1]

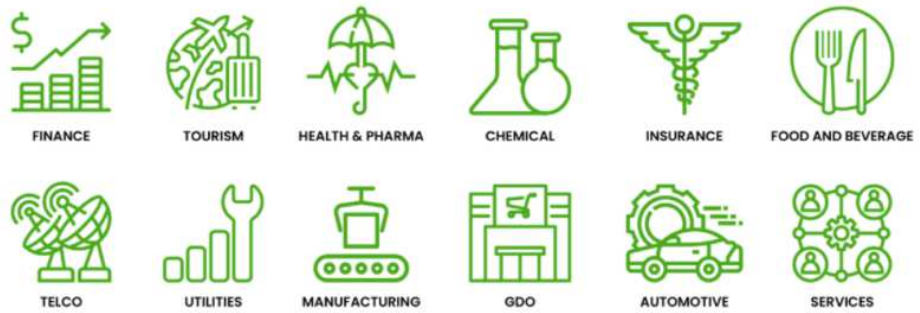


Figura 1.2: Settori in cui opera *Omicron Consulting*

L'organizzazione dell'azienda prevede la presenza di diversi dipartimenti interni, ognuno dedicato a diverse aree di competenza. In particolare, la sezione in cui ho avuto modo di lavorare è stata *Advanced Analytics (AA)*. Questa divisione si occupa di *Data Science* orientata soprattutto alla conoscenza delle dinamiche mercato e implementazioni, guidate dai dati, di strumenti utili all'organizzazione e alla strategia delle aziende. All'interno di questa divisione si lavora con grandi moli di dati (*Big Data*) e di conseguenza vengono utilizzati strumenti legati al *Machine Learning*, all'intelligenza artificiale, alla statistica e a sistemi di analitica avanzata. Le soluzioni di *Advanced Analytics* esaminano sia i dati storici aziendali che i dati esterni all'azienda, strutturati e non strutturati, concentrandosi sulla previsione di eventi e comportamenti futuri, disegnando scenari *"what-if"* per anticipare i cambiamenti e comprendere gli effetti a breve e lungo termine delle strategie aziendali. [2]

[1] Omicron Consulting, `https://www.omicronconsulting.it/approccio-e-mercati/`

[2] Omicron Consulting BI & Advanced Analytics, `https://www.omicronconsulting.it/bi-advanced-analytics/`

1.2 Modalità di lavoro

L'attività di stage è stata svolta dal 27 settembre al 26 novembre 2021, per un totale di 350 ore. Ho lavorato interamente in *smart working*, modalità che si è rivelata necessaria sia per l'emergenza pandemica di Covid-19, sia per motivi logistici, in quanto il *team* di cui ho fatto parte opera nella sede di Torino.
La modalità di lavoro a distanza non mi ha però impedito di relazionarmi e integrarmi pienamente con il gruppo di lavoro, composto da un totale di dieci persone.

Il team di lavoro era organizzato con modalità di lavoro *Agile*, che prevedono una serie di rituali e incontri cadenzati regolarmente per verificare l'avanzamento i progressi del progetto.
Ogni giornata lavorativa si apriva con un *Daily Meeting*, che rappresentava un'occasione per esporre a tutto il gruppo ciò che era stato fatto nel giorno precedente e, eventualmente, evidenziare le criticità riscontrate. Ogni due settimane erano inoltre previsti lo *Sprint Review* e lo *Sprint Planning*, riunioni nelle quali veniva fatto il punto della situazione sulle attività svolte fino a quel momento e si discutevano le fasi su cui concentrarsi per le due settimane successive.

Come accennato in precedenza, la procedura di raccolta dei dati si è limitata a due social, Twitter e Reddit. In particolare, per la prima fase del lavoro, io mi sono concentrata sui dati da Twitter. A partire dalla fase successiva di normalizzazione dei dati, mi è stato affiancato un altro stagista, con cui ho lavorato fino alla fase conclusiva dell'esperienza di stage.

1.3 Strumenti e piattaforme utilizzati

Durante tutta l'attività di stage ho lavorato con un computer aziendale, un *HP 250 G6 Notebook PC*, con sistema operativo *Windows 10*.
I programmi utilizzati sono stati:

- **Applicazioni di *Microsoft 365*** in particolare *Teams*
- ***RStudio*** (versione 4.1.1)

- ***Microsoft SQL Server Management Studio*** (versione v18.10)
- ***SourceTree*** (versione 3.4.6)

Le applicazioni web utilizzate sono state:

- ***Jupyter Notebook*** (versione 6.3.0)
- ***GitLab***

I linguaggi di programmazione utilizzati sono stati:

- ***R*** (su *RStudio*)
- ***Python*** (su *Jupyter Notebook*, con la versione Python 3.8.8)

Capitolo 2

Attività svolte

Viene ora presentata la procedura di acquisizione, analisi e interpretazione dei dati costruita durante l'esprienza di stage. Avendo personalmente lavorato sull'acquisizione dei dati da Twitter, il codice e tutti i risultati che seguono, inseriti come esempi, provengono da questo social network.

È però importante notare che la procedura implementata è una procedura standard che, ad eccezione della prima fase di raccolta dei dati, è valida e funzionante anche per Reddit. La normalizzazione è stata pensata e realizzata cercando le variabili in comune tra i due social, mentre le fasi successive vengono effettuate qualsiasi sia il dataframe dato in input.

Le fasi principali del lavoro sono individuate da:

- L'accesso alle *API* di Twitter;
- La creazione del dataframe con i dati di Twitter;
- L'esplorazione del dataset;
- La creazione del dataframe unico con i dati di Twitter e di Reddit;
- L'analisi testuale;
- La *Sentiment Analysis*;
- I risultati.

2.1 Accesso alle API di Twitter

Twitter dà la possibilità a utenti, società e programmatori di accedere a un numero elevato di dati pubblici tramite un servizio di *API*. Le **API** (***Application Programming Interface***) sono una serie di procedure, protocolli e strumenti che permettono alle applicazioni software di comunicare tra di loro e "scambiarsi" informazioni. In particolare, un'applicazione può chiamare un *endpoint*, cioè un elemento noto, codificato in modo univoco (come un URL), per consentire alle *API* di ottenere accesso alle risorse del server. [1]
Le *API* di Twitter includono diversi endpoint, a seconda dell'oggetto di interesse. Quelli utilizzati al fine del progetto rientrano nella categoria "Tweet e risposte", grazie ai quali gli sviluppatori sono in grado di accedere ai tweet cercando parole chiave specifiche. [2]

Per accedere alle *API* di Twitter è stato necessario seguire il percorso descritto dalla documentazione di Twitter presente nella sezione *Developer Platform*. Dopo aver richiesto e ottenuto l'autorizzazione per la creazione di un *"developer account"*, si è registrato un progetto e l'applicazione associata, generando anche le credenziali indispensabili per effettuare chiamate all'*API*. Queste credenziali comprendono l'username e la password dell'applicazione (*"API Key and Secret"*) e l'*Access Token*, un ulteriore codice che identifica l'utente proprietario dell'applicazione. [3]

Esistono diversi livelli di accesso alle *API* di Twitter, a seconda delle necessità di cui uno sviluppatore o un'azienda potrebbero aver bisogno. Per il progetto di stage si è deciso di usare la versione *Standard v1.1*, in quanto completamente gratuita e adeguata per il raggiungimento dell'obiettivo prefissato, anche se presenta qualche limite (di cui viene fatta menzione in seguito) rispetto ad altre versioni disponibili.

---

[1] The RapidAPI, *Endpoint - What is an API Endpoint?*, `https://rapidapi.com/blog/api-glossary/endpoint/`

[2] Centro Assistenza Twitter, *Informazioni sulle API di Twitter*, `https://help.twitter.com/it/rules-and-policies/twitter-api`

[3] Twitter Developer Platform, *How to get access to the Twitter API*, `https://developer.twitter.com/en/docs/twitter-api/getting-started/getting-access-to-the-twitter-api`

2.2 Creazione del dataframe con i dati di Twitter

Questa prima fase della procedura, che comprende la raccolta dei dati, l'arricchimento del dataframe con nuove informazioni e il filtraggio dei dati, è stata effettuata su *RStudio*.

Vengono presentati in seguito il codice prodotto e i risultati ottenuti.

2.2.1 Raccolta dei dati

Come prima cosa, è necessario caricare una libreria che permetta di accedere alle *API* di Twitter da *R*. La libreria che risulta essere più completa è `rtweet`, che mette a disposizione una vasta gamma di funzioni per l'acquisizione e l'organizzazione dei dati da Twitter.

```
> #Caricamento libreria
> library(rtweet)
```

Una delle funzioni principali presenti nella libreria `rtweet` è `search_tweets`, che permette la raccolta di tweet a partire da una o più parole chiave. Usando la versione *Standard v1.1* delle *API* di Twitter, `search_tweets` restituisce al massimo, ogni 15 minuti, 18 000 tweet scritti non oltre i 9 giorni precedenti al momento della ricerca. Per una ricerca più ampia, è possibile impostare l'argomento `retryonratelimit` della funzione uguale a `TRUE`. In questo modo, dopo circa 15 minuti, vengono raccolti ulteriori tweet recenti che vanno ad unirsi ai 18 000 iniziali.
Altri argomenti della funzione `search_tweets` sono: [4]

- **q**: è una stringa di caratteri che indica la/le parola/e chiave contenuta/e nei tweet da ricercare. Gli spazi equivalgono all'operatore booleano "AND" mentre, per cercare tweet che contengono almeno uno dei possibili termini, si separa ogni termine con lo spazio e l'operatore booleano "OR";

- **n**: indica il numero di tweet da ricercare. Di default, vale 100;

[4] Documentazione di R

- **include_rts**: indica se includere, nella ricerca, i tweet classificati come *retweet*. I *retweet* sono tweet ri-postati da altri utenti;
- **lang**: indica la lingua dei tweet da ricercare.

Si decide di limitare la ricerca ai tweet che contengono i termini "iPhone13", "iPhone 13" o "#iphone13". La funzione restituisce tutti i tweet più recenti scritti in lingua inglese, senza comprendere i retweet.

```
> #Raccolta dei dati
> prodotto<-'"iphone13" OR "iphone 13" OR "#iphone13"'
> tweets <- search_tweets(q=prodotto, n=18000,
   include_rts=F, retryonratelimit = T, lang="en")
> dim(tweets)

## [1] 16497    90
```

Viene creato il dataframe `tweets`, che ha come righe i tweet raccolti e come colonne le 90 variabili rese disponibili dalle *API*. Il numero di tweet raccolti risulta essere 16 497, inferiore ai 18 000 richiesti, ma questo significa che nei 9 giorni precedenti alla ricerca sono stati scritti solo 16 497 tweet contenenti le parole ricercate.

```
> colnames(tweets)

## [1] "user_id" "status_id"
## [3] "created_at" "screen_name"
## [5] "text" "source"
## [7] "display_text_width" "reply_to_status_id"
## [9] "reply_to_user_id" "reply_to_screen_name"
## [11] "is_quote" "is_retweet"
## [13] "favorite_count" "retweet_count"
## [15] "quote_count" "reply_count"
## [17] "hashtags" "symbols"
## [19] "urls_url" "urls_t.co"
## [21] "urls_expanded_url" "media_url"
## [23] "media_t.co" "media_expanded_url"
## [25] "media_type" "ext_media_url"
## [27] "ext_media_t.co" "ext_media_expanded_url"
## [29] "ext_media_type" "mentions_user_id"
## [31] "mentions_screen_name" "lang"
```

```
## [33] "quoted_status_id" "quoted_text"
## [35] "quoted_created_at" "quoted_source"
## [37] "quoted_favorite_count" "quoted_retweet_count"
## [39] "quoted_user_id" "quoted_screen_name"
## [41] "quoted_name" "quoted_followers_count"
## [43] "quoted_friends_count" "quoted_statuses_count"
## [45] "quoted_location" "quoted_description"
## [47] "quoted_verified" "retweet_status_id"
## [49] "retweet_text" "retweet_created_at"
## [51] "retweet_source" "retweet_favorite_count"
## [53] "retweet_retweet_count" "retweet_user_id"
## [55] "retweet_screen_name" "retweet_name"
## [57] "retweet_followers_count"
## [58] "retweet_friends_count"
## [59] "retweet_statuses_count" "retweet_location"
## [61] "retweet_description" "retweet_verified"
## [63] "place_url" "place_name"
## [65] "place_full_name" "place_type"
## [67] "country" "country_code"
## [69] "geo_coords" "coords_coords"
## [71] "bbox_coords" "status_url"
## [73] "name" "location"
## [75] "description" "url"
## [77] "protected" "followers_count"
## [79] "friends_count" "listed_count"
## [81] "statuses_count" "favourites_count"
## [83] "account_created_at" "verified"
## [85] "profile_url" "profile_expanded_url"
## [87] "account_lang" "profile_banner_url"
## [89] "profile_background_url" "profile_image_url"
```

Tra le variabili relative ai tweet, ce ne sono alcune particolarmente interessanti. Viene riportato il significato di quelle che vengono messe in risalto nel seguito della trattazione:

- ***user_id*** e ***status_id*** indicano il codice identificativo univoco de, rispettivamente, l'utente autore del tweet e del tweet stesso. Corrispondono a stringhe di numeri interi, con un numero variabile di cifre ciascuno;

```
> head(unique(tweets$status_id))

## [1] "1472865762595733505" "1470927510795735040"
## [3] "1472422298158202882" "1471372612357935104"
## [5] "1472022744598720517" "1469866559384424448"

> head(unique(tweets$user_id))

## [1] "1143266876471963648" "373854858"
## [3] "889302344868872192" "1023533797449580544"
## [5] "1126964830810923009" "1223569742364938240"
```

- ***screen_ name*** indica l'*username* (univoco) dell'autore del tweet;

```
> head(unique(tweets$screen_name))

## [1] "mochiiexiu" "Miss_DeeMeaner"
## [3] "Blogstiendientu" "Irunnia_"
## [5] "Mouler10" "newsgang2"
```

- ***created_ at*** indica la data e l'ora in cui il tweet è stato postato. Corrisponde a una stringa di caratteri;

```
> tweets$created_at[1:3]

## [1] "2021-12-20 09:45:36 UTC"
## [2] "2021-12-15 01:23:41 UTC"
## [3] "2021-12-19 04:23:26 UTC"
```

- ***text*** indica il contenuto scritto del tweet;

```
> head(unique(tweets$text))

## [1] "Iphone 13 cutie.\n\nChristmas gift please"
## [2] "@jiims @CryptoEthereal @shillempress
   @gemhostofficial @Wizardara @Razvi__Adil Iphone
    13 :) https://t.co/7cIlXFELVx"
## [3] "@Bwalya_II Oh my gosh, what series did you
    order? [...] One thing I've fought the
   temptation for is the iPhone 13 [...] "
```

```
## [4] "Win iPhone 13 pro Max and 200k for your
   business. [...]"
## [5] "iPhone 13 pro Max or 300k for your
   business?"
## [6] "@olatowealth1 12 pro Max better [...]"
```

- ***display_text_width*** indica il numero di caratteri del tweet. Non supera il valore di 280, anche se con alcune eccezioni;

```
> head(tweets$display_text_width)

## [1] 40 19 271 182 44 68
```

- ***reply_to_user_id*** e ***reply_to_status_id*** indicano, nel caso di *tweet risposta*, il codice identificativo univoco de, rispettivamente, l'utente autore del tweet "originale" e del tweet originale. Corrispondono a stringhe di numeri interi, con un numero variabile di cifre ciascuno. Valgono `NA` se il tweet a cui si riferiscono non è un *tweet risposta*;

```
> head(tweets$reply_to_status_id)

## [1] NA NA NA NA NA "1469632878568873986"
```

- ***is_quote*** e ***is_retweet*** indicano se il tweet a cui si riferiscono è, rispettivamente, un *tweet citazione* o un retweet. Hanno due possibili valori, `TRUE` o `FALSE`. Avendo eliminato dalla ricerca i retweet, ci si aspetta che `is_retweet` abbia sempre valore `FALSE`;

```
> head(tweets$is_quote)

## [1] FALSE FALSE FALSE FALSE FALSE FALSE

> head(tweets$is_retweet)

## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

- ***favorite_count***, ***retweet_count***, ***quote_count*** e ***reply_count*** indicano, rispettivamente, il numero di "preferiti" (o "mi piace"), di retweet, di citazioni e di risposte che ha ottenuto il tweet a cui si

riferiscono. Nella versione *Standard v1.1*, i valori di `quote_count` e `reply_count` non sono disponibili, quindi hanno sempre valore `NA`;

```
> head(tweets$favorite_count)

## [1] 0 0 0 0 0 1

> head(tweets$retweet_count)

## [1] 0 1 1 0 1 0

> head(tweets$quote_count)

## [1] NA NA NA NA NA NA

> head(tweets$reply_count)

## [1] NA NA NA NA NA NA
```

- ***lang*** indica la lingua in cui è stato scritto il tweet a cui si riferisce, rilevata dalla macchina con la codificazione *BCP 47*. Avendo ricercato solo tweet in inglese, ci si aspetta che abbia sempre valore `en`;

```
> head(tweets$lang)

## [1] "en" "en" "en" "en" "en" "en"
```

- ***followers_count***, ***friends_count***, ***listed_count***, ***statuses_count*** e ***favourites_count*** indicano, rispettivamente, il numero di *followers* (o "seguaci"), di *following* (o "seguiti"), di liste, di tweet e di "preferiti" (o di "mi piace") riferiti all'utente che ha scritto il tweet a cui si riferiscono.

```
> head(unique(tweets$followers_count))

## [1] 165 4920 217 269220 36 200

> head(unique(tweets$friends_count))

## [1] 377 1252 205 14164 89 1
```

```
> head(unique(tweets$listed_count))

## [1]  0 21 188  1  2 32

> head(unique(tweets$statuses_count))

## [1]  1414  575  144 184134  211 204549

> head(unique(tweets$favourites_count))

## [1]  3695 47956    86 56506   119    15
```

Ottenuti così i tweet contenenti le parole chiave, si è pensato che potesse essere interessante andare ad esplorare anche i tweet scritti in risposta a quelli trovati. Infatti, è plausibile che i commenti ad un tweet in cui viene menzionato il prodotto di riferimento, contengano opinioni o giudizi inerenti a quello, senza per forza includere nel testo la parola specifica.

Con la versione *Standard v1.1* delle *API* di Twitter, non esiste un modo veloce e automatico per recuperare tutte le risposte ad un determinato tweet. È stato dunque necessario ideare una procedura "indiretta", che ha però lo svantaggio di essere piuttosto lunga e computazionalmente onerosa.

Per limitare il più possibile questa difficoltà, si è deciso di ricercare le risposte solo per i tweet più popolari, ovvero quelli che hanno ottenuto almeno un "preferito" e un retweet. Si può pensare infatti che questi tweet siano quelli che hanno ottenuto più interazioni anche in termini di risposte.

```
> #Lavoro con i tweet che hanno ottenuto piu'
   popolarita'
> tweets_popolari <- tweets[tweets$favorite_count>
   0,]
> tweets_popolari <- tweets_popolari[tweets_
   popolari$retweet_count>0,]
```

```
> dim(tweets_popolari)

## [1] 2534   90
```

Vengono ricercate le risposte solo per 2 534 tweet, anziché per i 16 497 iniziali.

Un *tweet risposta* ha la particolarità di includere al suo interno la menzione (o il *tag*) dell'utente a cui viene risposto. Questa menzione è riconoscibile in quanto presenta l'*username* dell'utente preceduto da @. La procedura ideata consiste nell'andare a ricercare tutti i tweet recenti in cui l'utente è stato menzionato, per poi filtrare, tramite le variabili `reply_to_status_id` per i *tweet risposta* e `status_id` per i tweet originali, le effettive risposte ai tweet di nostro interesse.

```
> #Numero totale degli username
> screen_names <- unique(tweets_popolari$screen_
   name)
> n <- length(screen_names)
> n

## [1] 1915
```

Devono essere ricercati i tweet più recenti in cui sono stati menzionati i 1 915 utenti autori dei tweet più popolari.

I tweet vengono inseriti nella lista `z` che contiene, per ogni elemento, tutti i tweet (e le sue variabili relative) in cui l'utente x del vettore `screen_names` è stato menzionato.

```
> z <- list()
> for(i in 1:n) {
  z[[i]] <- search_tweets(q=paste0("@", screen_
     names[i]), retryonratelimit=TRUE)
}

> z[1:2]
```

```
## [[1]]
# A tibble: 5,698 x 90
   user_id             status_id  created_at
   <chr>               <chr>      <dttm>
1 1169757610855272448 147287594~ 2021-12-20 10:26:
   04
2 1169757610855272448 147260192~ 2021-12-19 16:17:
   13
# ... with 5,696 more rows, and 86 more variables:
    screen_name <chr>, text <chr>, source <chr>,
   ...

## [[2]]
# A tibble: 3 x 90
  user_id    status_id   created_at
  <chr>      <chr>       <dttm>
1 1616276844 1472866291~ 2021-12-20 09:47:43
2 1616276844 1472663112~ 2021-12-19 20:20:21
3 459646482  1472863123~ 2021-12-20 09:35:07
# ... with 87 more variables: screen_name <chr>,
   text <chr>, source <chr>, ...
```

La lista viene trasformata in un dataframe.

```
> mentions_data_frame <- as.data.frame(do.call(
   rbind, z)) #unione dei dati come dataframe

> dim(mentions_data_frame)

## [1] 1248024      90
```

Il dataframe `mentions_data_frame` è formato da 1 248 024 righe e 90 colonne (che corrispondono alle variabili date in output dalla funzione `search_tweets` viste in precedenza).

Non tutti i tweet dove viene menzionato un utente sono *tweet risposta*. Esistono anche casi in cui la menzione è presente, ma il tweet non è stato scritto come commento a un tweet precedente: questo accade ad esempio nel caso in cui un utente voglia iniziare una conversazione con un altro. Per questo è necessario fare una distinzione, resa possibile dalla variabile `reply_to_status_id`

(oppure, in egual modo, da `reply_to_user_id`): se si tratta di un *tweet risposta*, questa variabile contiene il codice univoco del tweet originale; altrimenti, contiene il valore `NA`.

```
> all_replies_data_frame <- subset(mentions_data_
   frame, !is.na(mentions_data_frame$reply_to_
   status_id))

> dim(all_replies_data_frame)

## [1] 495093     90
```

Il dataframe `all_replies_data_frame` contiene i dati relativi a 495 093 *tweet risposta*.

Vengono ora individuati i *tweet risposta* ai tweet originali contenenti le parole chiave ricercate. Per fare ciò viene fatto un confronto tra il valore della variabile `reply_to_status_id` nel dataframe appena creato e il valore di `status_id` nel dataframe contentente i tweet più popolari.

Si inseriscono in un vettore tutti i codici univoci (*ID*) trovati in comune.

```
> #ID in comune
> id_in_common <- Reduce(intersect, list(all_
   replies_data_frame$reply_to_status_id, tweets_
   popolari$status_id))

> m <- length(id_in_common)
> m

## [1] 815
```

Il vettore `id_in_common` ha lunghezza 815, che corrisponde al numero di tweet popolari per cui sono state trovate delle risposte. Le diverse risposte vengono inserite nella lista `r`, che ha come elementi i *tweet risposta* per ogni tweet originale.

```
> r <- list()
> for(i in 1:m) {
```

```
r[[i]] <- subset(all_replies_data_frame, all_
   replies_data_frame$reply_to_status_id==id_in_
   common[i])
}

> some_replies <- as.data.frame(do.call(rbind, r))
> dim(some_replies)

## [1] 10852    90
```

La lista `r` viene trasformata nel dataframe `some_replies`, che contiene dunque le 10 852 risposte ai tweet originali più popolari.

Si filtrano ulteriormente le risposte, in modo tale che anche esse siano tutte in lingua inglese (come i tweet originali).

```
> #Filtro per lingua inglese
> some_replies <- some_replies[some_replies$lang==
   "en",]
> dim(some_replies)

## [1] 6444   90
```

Infine, si unisce il dataframe `some_replies` con il dataframe `tweets`, sotto il nome di `Twitter.data`.

```
> #Unione dei tweet originali con tweet risposta
> Twitter.data <- rbind(tweets, some_replies)
> dim(Twitter.data)

## [1] 22941    90
```

Il dataframe a questo punto è quindi formato da 22 941 tweet, di cui 6 444 non contengono l'espressione "iPhone 13", ma è plausibile che si riferiscano comunque ad esso.

2.2.2 Creazione di nuove variabili

A questo punto, ottenuti i tweet su cui lavorare, è necessario manipolare il dataframe in modo tale che, da una parte, contenga quante

più informazioni possibili con i dati a disposizione, e dall'altra che sia coerente con il tema di riferimento.

Vengono aggiunte delle nuove variabili, che vanno ad aggiungersi (o a sostituire) quelle già rese disponibili dalle *API* di Twitter:

- ***content_date*** e ***content_time*** indicano la data e l'ora in cui è stato scritto il tweet a cui si riferiscono. Vanno a sostituire la variabile `created_at`, che conteneva, in un'unica stringa, entrambe le informazioni;

```
> x <- format.Date(Twitter.data$created_at)
> y = t(as.data.frame(strsplit(as.character(x),' ')
   )))
> row.names(y) = NULL
> Twitter.data$content_date <- y[,1]
> Twitter.data$content_time <- y[,2]

> head(Twitter.data$content_date)

## [1] "2021-12-20" "2021-12-15" "2021-12-19" "202
   1-12-16" "2021-12-18" "2021-12-12"

> head(Twitter.data$content_time)

## [1] "09:45:36" "01:23:41" "04:23:26" "06:52:22"
    "01:55:45" "03:07:51"

> #Rimozione della variabile originale created_at
> Twitter.data <- subset(Twitter.data, select =
      -created_at)
> dim(Twitter.data)

## [1] 22941    91
```

- ***content_date*** e ***content_time*** indicano la data e l'ora in cui è stato creato l'account dell'autore del tweet a cui si riferiscono. Vanno a sostituire la variabile `account_created_at`, che conteneva, in un'unica stringa, entrambe le informazioni;

```
> x <- format.Date(Twitter.data$account_created_at
   )
> y = t(as.data.frame(strsplit(as.character(x),'␣'
   )))
> row.names(y) = NULL
> Twitter.data$account_date <- y[,1]
> Twitter.data$account_time <- y[,2]

> head(Twitter.data$account_date)

## [1] "2019-06-24" "2019-06-24" "2019-06-24" "201
   9-06-24" "2019-06-24" "2019-06-24"

> head(Twitter.data$account_time)

## [1] "21:17:18" "21:17:18" "21:17:18" "21:17:18"
    "21:17:18" "21:17:18"

> #Rimozione della variabile originale account_
   created_at
> Twitter.data <- subset(Twitter.data, select =
       -account_created_at)
> dim(Twitter.data)

## [1] 22941    92
```

- ***ripetizioni*** indica quante volte uno stesso tweet è presente all'interno del dataset. La variabile restituisce `1` se il tweet è unico, mentre restituisce `n` *(> 1)* se il tweet è ripetuto n volte;

```
> contatore_per_tweets <- function(tweets) {
  n <- length(tweets$text)
  CONT <- rep(1, n)
  for(i in 1:(n-1)) {
    for(j in (i+1):n) {
      if (identical(tweets$text[i], tweets$text[j
         ])) {
        CONT[i] <- CONT[i]+1
        CONT[j] <- CONT[j]+1
      }
```

```
    }
  }
  return (CONT)
}
> Twitter.data$ripetizioni <- contatore_per_tweets
   (Twitter.data)
> head(Twitter.data$ripetizioni)

## [1]  1 17  2  3  3  1

> dim(Twitter.data)

## [1] 22941    93
```

- **is_reply** indica se il tweet a cui fa riferimento è un *tweet risposta*. Ha due possibili valori, `TRUE` o `FALSE`;

```
> is_reply <- function(tweets) {
  n <- dim(tweets)[1]
  TF_risposta <- rep(NULL, n)
  for(i in 1:n) {
    if (is.na(tweets$reply_to_status_id[i]))
        {
            TF_risposta[i]=FALSE
        }
    else
    {
        TF_risposta[i]=TRUE}
    }
  return (TF_risposta)
}
> Twitter.data$is_reply <- is_reply(Twitter.data)
> dim(Twitter.data)

## [1] 22941    94
```

- **type_text** indica se il tweet a cui si riferisce è un *tweet citazione* (`quote`), un *tweet risposta* (`reply`), un retweet (`retweet`)

o un tweet semplice (`organic`). La variabile è stata costruita facendo riferimento ai valori delle variabili `is_reply`, `is_retweet` e `is_quote`;

```
> type_text <- function(tweets) {
  n <- dim(tweets)[1]
  tipo_testo <- rep(NULL, n)
  for(i in 1:n) {
    if (tweets$is_reply[i]==TRUE) {
      tipo_testo[i]="reply"
    }
    else if (tweets$is_retweet[i]==TRUE) {
      tipo_testo[i]="retweet"
    }
    else if (tweets$is_quote[i]==TRUE) {
      tipo_testo[i]="quote"
    }
    else {
      tipo_testo[i]="organic"
    }
  }
  return (tipo_testo)
}
> Twitter.data$type_text <-type_text(Twitter.data)
> dim(Twitter.data[Twitter.data$type_text=="
   organic",])

## [1] 10641    95

> dim(Twitter.data[Twitter.data$type_text=="quote"
   ,])

## [1] 737  95

> dim(Twitter.data[Twitter.data$type_text=="reply"
   ,])

## [1] 11563    95
```

```
> dim(Twitter.data)

## [1] 22941    95
```

- ***product_user_count*** indica quante volte uno stesso utente ha twittato sull'argomento (ovvero, quante volte compare un tweet scritto dall'utente all'interno del dataset). La variabile restituisce `1` se l'utente ha scritto solo un tweet, mentre restituisce `n` *(> 1)* se l'utente ha scritto n tweet presenti nel dataset;

```
> contatore_per_account <- function(tweets) {
  n <- length(tweets$screen_name)
  CONT <- rep(1, n)
  for(i in 1:(n-1)) {
    for(j in (i+1):n) {
      if (identical(tweets$screen_name[i], tweets$
          screen_name[j])) {
        CONT[i] <- CONT[i]+1
        CONT[j] <- CONT[j]+1
      }
    }
  }
  return (CONT)
}
> Twitter.data$product_user_count <- contatore_per
   _account(Twitter.data)
> head(Twitter.data$product_user_count)

## [1] 7 7 7 7 7 7

> dim(Twitter.data)

## [1] 22941    96
```

- ***social_media*** indica il social network a cui stiamo facendo riferimento. Ovviamente, in questo caso, il suo valore corrisponde sempre alla stringa di caratteri "Twitter".

```
> #Variabile che indica il social network
> Twitter.data$social_media <- "Twitter"
```

```
> dim(Twitter.data)

## [1] 22941    97
```

Sono state quindi inserite 9 nuove variabili per arricchire l'informazione disponibile. Molte di queste, come si vedrà successivamente, sono in comune con le variabili raccolte sui dati da Reddit.

2.2.3 Filtraggio dei dati

Come detto in precedenza, è necessario che i tweet trovati siano inerenti con il *product* di riferimento. Per garantire una certa coerenza nel dataset, si è deciso di filtrare i dati.

Questo è stato fatto costruendo un "dizionario" (esterno al programma) che contiene diverse parole che ci si aspetta di trovare associate ad "iPhone 13". Sono stati dunque eliminati i tweet che non contenevano alcuna di tali parole.

Questo passaggio ha come svantaggio l'eliminazione di informazioni potenzialmente utili, in quanto cancella di fatto numerosi tweet, ma dall'altra parte assicura che i tweet del dataset definitivo siano relativi al tema d'interesse.

Alcune delle parole inserite nel vocabolario sono:

- pro
- max
- apple
- android
- ios
- media
- [...]

Si sottolinea che il dizionario deve essere costruito ogni qualvolta si desideri attuare la procedura per un *product* di riferimento diverso, e ne presuppone una certa conoscenza. Inoltre, il dizionario può

essere modificato e reso più o meno ricco di termini specifici, a seconda del grado di "pertinenza" che si è in grado di accettare relativamente al tema scelto.

```
> #Vocabolario con parole che possono essere
   associate con il prodotto
> library(purrr)
> txt <- purrr::as_vector(read.table(file.choose()
   , sep="\n"))

> #Si tengono solo i tweet che contengono
   determinate parole chiave
> x1 <- Reduce('|', lapply(txt, function(x) grepl(
   x, Twitter.data$text)))
> Twitter.data <- Twitter.data[x1,,drop=FALSE]
> dim(Twitter.data)

## [1] 11148    97
```

L'utilizzo del dizionario ha comportato l'eliminazione di 12 485 tweet.

L'ultima modifica al dataset consiste nella rimozione dei tweet presenti più volte. Questo viene attuato per evitare successive distorsioni nelle analisi, ma anche per salvaguardare la veridicità dei dati dalla presenza di *tweet spam* scritti da *bot* o da account inautentici, ovvero di messaggi ripetuti a fini pubblicitari o di disturbo.

```
> #Eliminazione dei tweet ripetuti
> Twitter.data_ripetuti <- subset(Twitter.data,
   Twitter.data$ripetizioni>1)
> Twitter.data_ripetuti_unici <-  Twitter.data_
   ripetuti[!duplicated(Twitter.data_ripetuti$text
   ),]
> Twitter.data_non_ripetuti <- subset(Twitter.data
   , Twitter.data$ripetizioni==1)
> Twitter.data <- rbind(Twitter.data_ripetuti_
   unici, Twitter.data_non_ripetuti)
```

```
> dim(Twitter.data)

## [1] 10456    97
```

Il dataset definitivo è composto dunque da 10 456 tweet (osservazioni), a cui fanno riferimento 97 variabili.

2.3 Esplorazione del dataset

Ottenuta in questo modo una solida base per le analisi, è d'interesse andare a valutare il contenuto del dataset. Anche questa fase del lavoro è stata svolta su *RStudio*.

Il dataset definitivo contiene 422 *tweet citazione* e 4198 *tweet risposta*, mentre i restanti 5836 sono tweet semplici ed indipendenti.

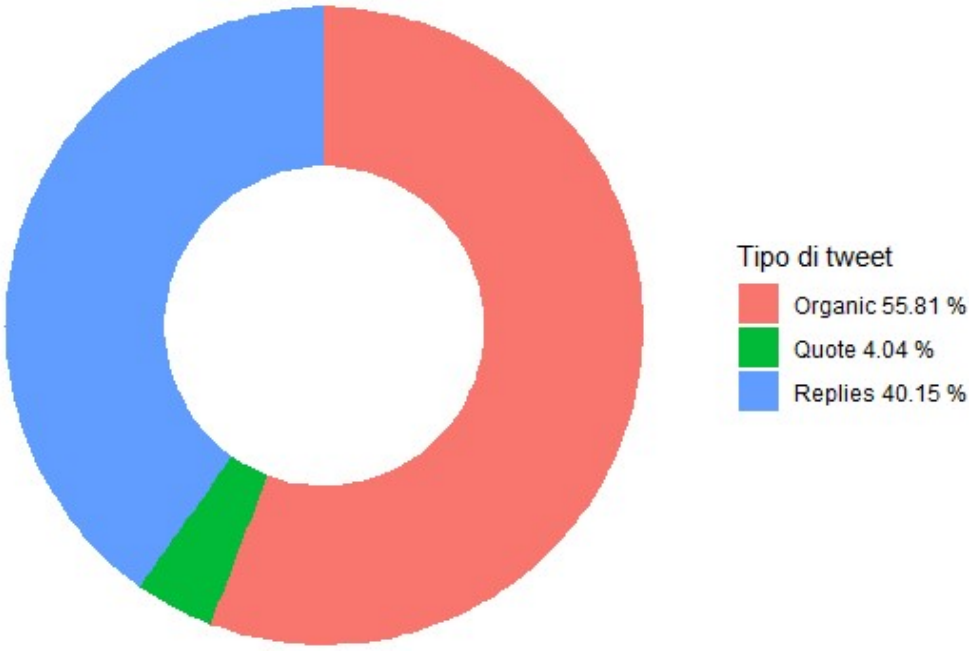


Figura 2.1: *Donut Chart* rappresentante le frequenze percentuali dei tweet presenti nel dataset, divisi per tipo

In un primo momento, viene esplorata la relazione tra le variabili, per poi procedere con l'*Analisi delle Componenti Principali (PCA)*.

2.3.1 Individuazione delle variabili numeriche

Per valutare le relazioni presenti tra le variabili del dataset e attuare l'*Analisi delle Componenti Principali*, è doveroso considerare solo le variabili numeriche. Si vanno ad individuare.

```
> #Variabili numeriche
> library(dplyr)
> TwitterDataNumeric <- dplyr::select_if(Twitter.
   data, is.numeric)

> dim(TwitterDataNumeric)

## [1] 10456    22
```

Le variabili numeriche del dataset sono 22.

Un aspetto su cui fare attenzione è la presenza di valori nulli. Dato che questi possono generare diversi problemi nelle analisi, è opportuno valutare fin dal principio come elaborarli.

Per esempio, tra le 22 variabili numeriche trovate, ce ne sono alcune che presentano sempre valori nulli. Ne sono un esempio `quote_count` e `reply_count` che, per un limite della versione delle *API* utilizzata, valgono sempre `NA`, o tutte le variabili relative ai retweet, in quanto questi non sono stati compresi dal principio nella ricerca. Si decide quindi di eliminare queste variabili, che non danno alcuna informazione.

```
> #Si eliminano variabili che hanno sempre valore
   nullo
> TwitterDataNumeric2 <- dplyr::select(
   TwitterDataNumeric, -c(quote_count, reply_count
   , retweet_favorite_count, retweet_retweet_count
   , retweet_followers_count, retweet_friends_
   count, retweet_statuses_count))

> dim(TwitterDataNumeric2)

## [1] 10456    15
```

Il dataset è stato ridotto a 15 variabili. Tuttavia, anche le variabili che sono state mantenute presentano alcuni valori nulli. Si valuta quindi, per ogni variabile, il rapporto tra il numero di valori mancanti e i valori totali: se questo è sufficientemente basso, le rispettive variabili possono essere tenute in considerazione. Come soglia si sceglie 0.1, vengono quindi tenute nel dataframe le variabili per cui i valori nulli sono meno del 10% dei valori totali.

```
> densita <- list()
> for(i in 1:dim(TwitterDataNumeric2)[2]) {
densita[i] <- sum(is.na(TwitterDataNumeric2)[,i])/
   dim(TwitterDataNumeric2)[1]
}
> densita

## [[1]]
## [1] 0

## [[2]]
## [1] 0

## [[3]]
## [1] 0

## [[4]]
## [1] 0.9592578

## [[5]]
## [1] 0.9592578

## [[6]]
## [1] 0.9592578

## [[7]]
## [1] 0.9592578

## [[8]]
## [1] 0.9592578
```

```
## [[9]]
## [1] 0

## [[10]]
## [1] 0

## [[11]]
## [1] 0

## [[12]]
## [1] 0

## [[13]]
## [1] 0

## [[14]]
## [1] 0

## [[15]]
## [1] 0
```

I valori dei rapporti `densita` ottenuti per le variabili `quoted_favorite_count`, `quoted_retweet_count`, `quoted_followers_count`, `quoted_friends_count` e `quoted_statuses_count` sono di numero maggiore allo 0.90. Questo risultato era prevedibile in quanto si tratta delle variabili relative ai *tweet citazione* che rappresentano, come visto in precedenza, solo il 4% del totale delle osservazioni. Anche queste variabili vengono dunque eliminate, in quanto poco informative.

```
> index=rep(0, 15)
> for (i in 1:15) {
  if (densita[[i]][1] > 0.1) {
    index[i]=i
  }
}
> TwitterDataNumeric3 <- (TwitterDataNumeric2[,-
   index])

> dim(TwitterDataNumeric3)
```

```
## [1] 10456    10
```

Le variabili numeriche su cui verranno svolte le analisi sono 10. Viene fatto notare che è un numero molto piccolo rispetto a quello delle variabili presenti nel dataframe originale.

2.3.2 Analisi delle relazioni tra le variabili

Si compie una prima analisi esplorativa sulle variabili, andando ad osservare le statistiche principali con la funzione `summary` e i grafici *boxplot*.

```
> summary(TwitterDataNumeric3)

## display_text_width      favorite_count
   Min.   :  4.0           Min.   :   0.000
   1st Qu.: 51.0           1st Qu.:   0.000
   Median :105.0           Median :   0.000
   Mean   :126.4           Mean   :   7.167
   3rd Qu.:198.0           3rd Qu.:   1.000
   Max.   :303.0           Max.   :7557.000

## retweet_count           followers_count
   Min.   :   0.000        Min.   :       0
   1st Qu.:   0.000        1st Qu.:      38
   Median :   0.000        Median :     236
   Mean   :   1.889        Mean   :    7973
   3rd Qu.:   0.000        3rd Qu.:    1174
   Max.   :5147.000        Max.   :10256719

## friends_count           listed_count
   Min.   :     0.0        Min.   :    0.00
   1st Qu.:    93.0        1st Qu.:    0.00
   Median :   338.5        Median :    1.00
   Mean   :  1155.5        Mean   :   51.36
   3rd Qu.:   993.2        3rd Qu.:    6.00
   Max.   :159344.0        Max.   :89304.00

## statuses_count          favourites_count
```

```
   Min.    :      1.0        Min.    :      0
   1st Qu.:    759.8        1st Qu.:    260
   Median :   5793.0        Median :   2908
   Mean   :  30003.1        Mean   :  16807
   3rd Qu.:  25084.0        3rd Qu.:  15335
   Max.   :1951690.0        Max.   : 829615

## ripetizioni              product_user_count
   Min.   : 1.000           Min.   :  1.00
   1st Qu.: 1.000           1st Qu.:  1.00
   Median : 1.000           Median :  1.00
   Mean   : 1.066           Mean   : 15.14
   3rd Qu.: 1.000           3rd Qu.:  2.00
   Max.   :70.000           Max.   :235.00
```

```
> #Boxplot
> par(mfrow=c(1,5))
> par(cex.main=0.7)
> for(i in 1:dim(TwitterDataNumeric3)[2]) {
  boxplot(TwitterDataNumeric3[,i], main=paste0(i,
     ". Boxplot - ", colnames(TwitterDataNumeric3)
     [i]))
}
```

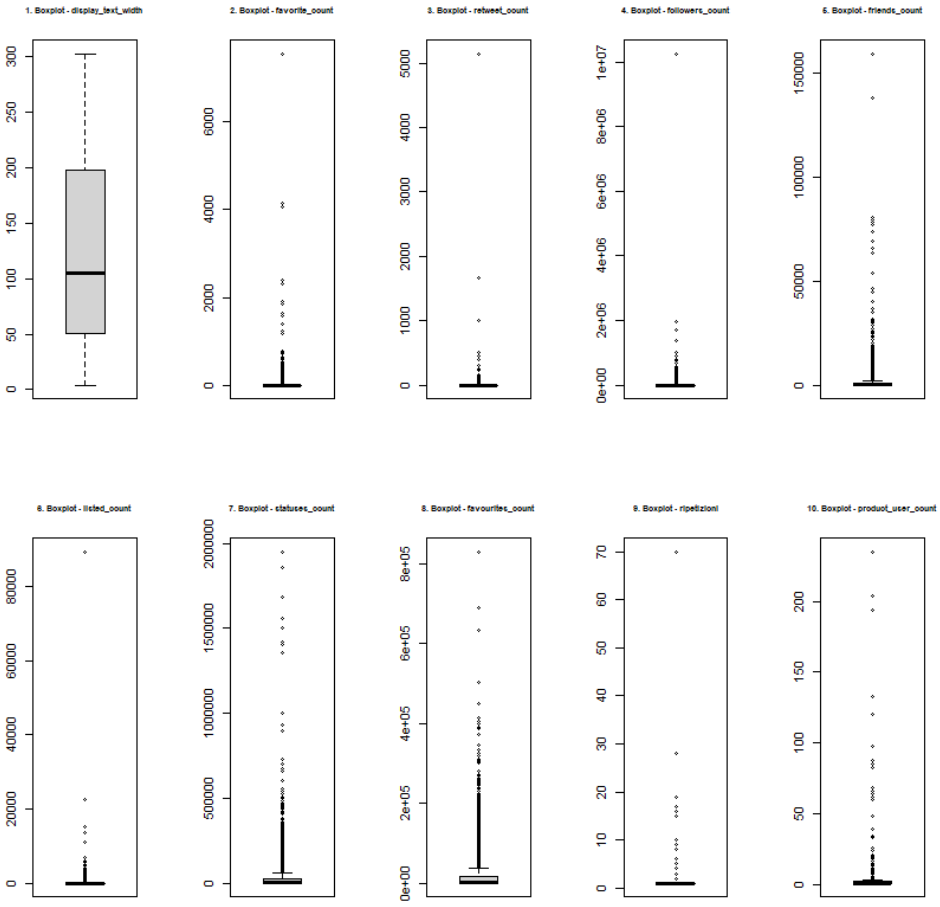


Figura 2.2: Boxplot relativi alle variabili numeriche

Osservando le statistiche e i boxplot relativi alle variabili, si nota che si distribuiscono tutte su scale molto diverse tra di loro.

Per quasi tutte le variabili (con eccezione di `display_text_width`) si nota che la media è maggiore del terzo quartile: questo sta ad indicare che i valori anomali vanno ad alterare l'attendibilità della (poco robusta) media come misura sintetica sull'andamento del fenomeno. Tutte le variabili sono caratterizzate da asimmetria verso destra. Per tutti i boxplot sono visibili diverse osservazioni anomale, tranne che per la variabile `display_text_width`.

`display_text_width` è anche l'unica variabile per cui esiste un limite superiore. Essa infatti indica il numero di caratteri di un tweet e, per definizione, un tweet non può superare una certa soglia di caratteri (280). Tuttavia, può succedere che un tweet superi

di poco questo valore, in quanto esistono caratteri *Unicode* che vengono contati come *più caratteri* (è il caso, per esempio, di alcune *emoji*) [5]. In ogni caso, il numero di caratteri in un tweet non potrà mai superare il valore di 300/350.

Si vanno ora ad analizzare i grafici di dispersione e la matrice di correlazione per verificare la relazione tra le diverse variabili.

```
> #Relazione tra le variabili
> pairs(TwitterDataNumeric3, col=2, gap=0.01)
```

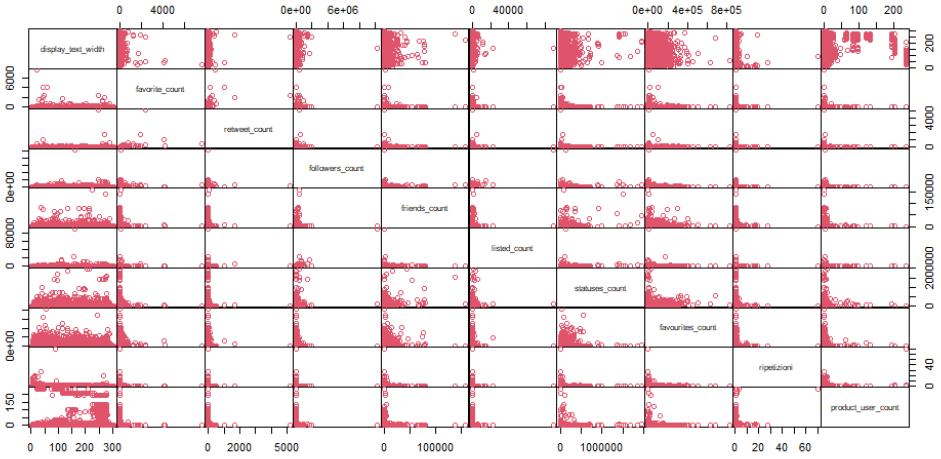


Figura 2.3: Grafici di dispersione delle variabili numeriche

```
#Matrice di correlazione
> round(cor(TwitterDataNumeric3),5)

##                  display_text_width     favorite_count
## display_text_width           1.00000           -0.00434
## favorite_count              -0.00434            1.00000
## retweet_count                0.02583            0.42552
## followers_count              0.03738            0.07124
## friends_count                0.04390            0.02322
## listed_count                 0.02327            0.02700
## statuses_count               0.09941            0.00345
```

[5]Twitter Developer Platform, *Counting characters*, `https://developer.twitter.com/en/docs/counting-characters`

```
## favourites_count     -0.03254           0.03341
## ripetizioni          -0.04331          -0.00348
## product_user_count    0.28992          -0.01812

##                retweet_count    followers_count
## display_text_width    0.02583           0.03738
## favorite_count        0.42552           0.07124
## retweet_count         1.00000           0.05243
## followers_count       0.05243           1.00000
## friends_count         0.01519           0.07848
## listed_count          0.01976           0.88036
## statuses_count        0.00401           0.17540
## favourites_count      0.02139           0.01399
## ripetizioni          -0.00227          -0.00429
## product_user_count   -0.00706          -0.01111

##                friends_count       listed_count
## display_text_width    0.04390           0.02327
## favorite_count        0.02322           0.02700
## retweet_count         0.01519           0.01976
## followers_count       0.07848           0.88036
## friends_count         1.00000           0.03011
## listed_count          0.03011           1.00000
## statuses_count        0.34492           0.11547
## favourites_count      0.22827           0.00833
## ripetizioni          -0.00239          -0.00363
## product_user_count    0.05090          -0.00622

##               statuses_count   favourites_count
## display_text_width    0.09941          -0.03254
## favorite_count        0.00345           0.03341
## retweet_count         0.00401           0.02139
## followers_count       0.17540           0.01399
## friends_count         0.34492           0.22827
## listed_count          0.11547           0.00833
## statuses_count        1.00000           0.22270
## favourites_count      0.22270           1.00000
## ripetizioni          -0.01374          -0.01295
## product_user_count    0.11071          -0.03824
```

```
                    ripetizioni product_user_count
## display_text_width  -0.04331            0.28991
## favorite_count      -0.00348           -0.01812
## retweet_count       -0.00227           -0.00706
## followers_count     -0.00429           -0.01111
## friends_count       -0.00239            0.05090
## listed_count        -0.00363           -0.00622
## statuses_count      -0.01374            0.11071
## favourites_count    -0.01295           -0.03824
## ripetizioni          1.00000            0.02646
## product_user_count   0.02646            1.00000
```

I diagrammi di dispersione sono difficilmente interpretabili, in quanto il numero di osservazioni è molto grande, ma con l'aiuto della matrice di correlazione si possono notare alcune associazioni positive, in particolare quelle tra `favorite_count` e `retweet_count` (0.425) e tra `followers_count` e `listed_count` (0.880). Molti coefficienti di correlazioni sono comunque molto bassi, ad indicare che tra alcune variabili c'è correlazione quasi nulla.

2.3.3 Analisi delle componenti principali

L'**Analisi delle Componenti Principali** (**PCA**) è una tecnica statistica che si pone come obiettivo quello di ridurre la dimensionalità di un insieme di dati e di interpretarlo. In particolare, a partire da un vettore di p variabili, si possono trovare altre k variabili incorrelate tra di loro (con $k<p$) che sono ugualmente in grado di spiegare la matrice di covarianza delle p variabili originali. Le nuove k variabili sono dette **componenti principali** e sono delle trasformazioni lineari delle p variabili iniziali, che possono sostituirle per formare un nuovo dataset con k variabili e lo stesso numero di osservazioni. [6]

Per attuare su *RStudio* l'*Analisi delle Componenti Principali* si utilizza il comando `prcomp`. Il numero di componenti principali da

[6] Johnson R.A.; Wichern D.W. *Applied Multivariate Statistical Analysis.* Pearson Education Limited, 2014, Sixth Edition

tenere si sceglie considerando la proporzione cumulata di varianza spiegata, che deve raggiungere almeno l'80%. È riportato anche lo *scree plot* corrispondente, cioè il grafico degli autovalori, dove viene individuato il "punto di gomito" che dà un ulteriore suggerimento empirico sul numero di componenti principali da prendere in esame. Questo corrisponde al punto dove gli autovalori raffigurati dal grafico iniziano ad essere relativamente piccoli e quasi tutti dello stesso valore.

```
> #PCA con dati originali
> pc <- prcomp(TwitterDataNumeric3)
> summary(pc)

## Importance of components:
##                             PC1           PC2
## Standard deviation     1.211e+05     7.994e+04
## Proportion of Variance 6.538e-01     2.849e-01
## Cumulative Proportion  6.538e-01     9.388e-01

##                             PC3           PC4
## Standard deviation     3.687e+04     3.618e+03
## Proportion of Variance 6.063e-02     5.800e-04
## Cumulative Proportion  9.994e-01     1.000e+00

##                             PC5    PC6     PC7
## Standard deviation     460.37587  112.3   87.55
## Proportion of Variance   0.00001    0.0    0.00
## Cumulative Proportion    1.00000    1.0    1.00

##                             PC8    PC9    PC10
## Standard deviation         48.33  40.54   0.884
## Proportion of Variance      0.00   0.00   0.000
## Cumulative Proportion       1.00   1.00   1.000
```

```
> #Numero di componenti principali da tenere
> TwitterPCANumber <- 0
> for(i in 1:dim(pc$x)[2]) {
  if (summary(pc)$importance["Cumulative␣
     Proportion",i] <= 0.9) {
    TwitterPCANumber <- TwitterPCANumber+1
```

```
  }
}
> TwitterPCANumber

## [1] 2
```

```
> #Scree plot
> plot(pc, type="l", main="Scree␣plot␣(dati␣
   originali)")
```

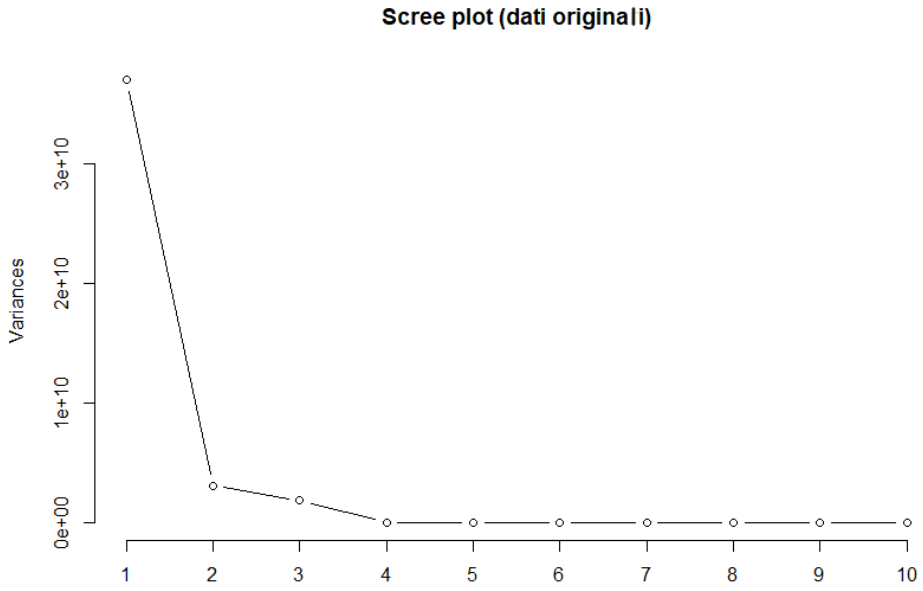


Figura 2.4: Scree plot dei dati originali

I risultati del `summary` indicano che la prima componente principale, da sola, spiega l'88.37% della varianza totale, mentre le prime due insieme ne spiegano quasi la totalità (95.65%). Anche lo *scree plot* conferma questi risultati.

Come si è visto in precedenza, le variabili prese in considerazione presentano scale e valori molto diversi tra di loro. È opportuno valutare se lavorare con i dati originali possa dare una giusta interpretazione alle componenti principali trovate, oppure se è conveniente applicare la *PCA* sui dati standardizzati.

```
> #Matrice di covarianza
```

```
> round(var(TwitterDataNumeric3),3)

##                  display_text_width     favorite_count
## display_text_width          7582.585            -41.591
## favorite_count              -41.591          12102.973
## retweet_count               123.521           2571.097
## followers_count          389190.106         936937.457
## friends_count             14951.053           9990.494
## listed_count               1977.174           2898.602
## statuses_count           706729.524          30984.620
## favourites_count       -107976.541         140059.748
## ripetizioni              -3.341              -0.339
## product_user_count         1089.495            -86.049

##                       retweet_count    followers_count
## display_text_width          123.521              3.892
## favorite_count             2571.097              9.369
## retweet_count              3016.542              3.443
## followers_count          344294.271              1.429
## friends_count              3262.806              3.669
## listed_count               1059.031              1.027
## statuses_count            17967.513              1.712
## favourites_count          44775.495              6.375
## ripetizioni                 -0.111             -4.541
## product_user_count         -16.732             -5.734

##                       friends_count       listed_count
## display_text_width       14951.053           1977.174
## favorite_count            9990.494           2898.602
## retweet_count             3262.806           1059.031
## followers_count       36693767.395      102690725.199
## friends_count         15293576.641         114882.524
## listed_count            114882.524         952011.567
## statuses_count       110125588.768        9198353.911
## favourites_count      34016097.740         309629.111
## ripetizioni              -8.267              -3.135
## product_user_count        8589.865            -262.010

##                     statuses_count   favourites_count
```

```
## display_text_width     7.067              -1.080
## favorite_count         3.098               1.401
## retweet_count          1.797               4.477
## followers_count        1.712               6.375
## friends_count          1.101               3.402
## listed_count           9.198               3.096
## statuses_count         6.666               6.928
## favourites_count       6.928               1.452
## ripetizioni           -9.934              -4.369
## product_user_count     3.901              -6.289

##                   ripetizioni product_user_count
## display_text_width    -3.341           1089.495
## favorite_count        -0.339            -86.049
## retweet_count         -0.111            -16.732
## followers_count     -454.098         -57342.827
## friends_count         -8.267           8589.865
## listed_count          -3.135            -262.010
## statuses_count      -993.373         390082.635
## favourites_count    -436.944         -62889.981
## ripetizioni            0.785              1.011
## product_user_count     1.011           1862.472
```

Anche la matrice di covarianza conferma la forte differenza tra le variabili analizzate. Si attua quindi l'*Analisi delle Componenti Principali* sui dati standardizzati, che sono stati costruiti tramite la matrice di correlazione.

```
> #PCA con dati standardizzati
> pc2 <- prcomp(TwitterDataNumeric3, scale.=TRUE)
> summary(pc2)

## Importance of components:
                            PC1      PC2      PC3
## Standard deviation     1.4114   1.2239   1.1904
## Proportion of Variance 0.1992   0.1498   0.1417
## Cumulative Proportion  0.1992   0.3490   0.4907

##                            PC4      PC5      PC6
## Standard deviation     1.1312   1.0039  0.88101
```

```
## Proportion of Variance 0.1280   0.1008   0.07762
## Cumulative Proportion  0.6187   0.7194   0.79706

##                             PC7          PC8
## Standard deviation          0.83707      0.80024
## Proportion of Variance      0.07007      0.06404
## Cumulative Proportion       0.86713      0.93117

##                             PC9          PC10
## Standard deviation          0.75679      0.34003
## Proportion of Variance      0.05727      0.01156
## Cumulative Proportion       0.98844      1.00000
```

```
> #Numero di componenti principali da tenere
> TwitterPCANumber <- 0
> for(i in 1:dim(pc2$x)[2]) {
  if (summary(pc2)$importance["Cumulative␣
     Proportion",i] <= 0.9) {
    TwitterPCANumber <- TwitterPCANumber+1
  }
}
> TwitterPCANumber

## [1] 7
```

```
> #Scree plot
> plot(pc2, type="l", main="Scree␣plot␣(dati␣
   standardizzati)")
```

Nel caso della *PCA* per i dati standardizzati, il numero di componenti principali da tenere in considerazione è 7. Infatti, tenendo le prime 6 componenti principali si raggiunge il valore della varianza spiegata cumulata di 0.797, mentre considerando anche la settima le componenti principali spiegano l'86% della varianza totale. Nello *scree plot* il "punto di gomito" non è molto evidente.

Per valutare a quali variabili si riferiscono le componenti principali, si costruisce la matrice di rotazione.

```
> #Matrice di rotazione
```

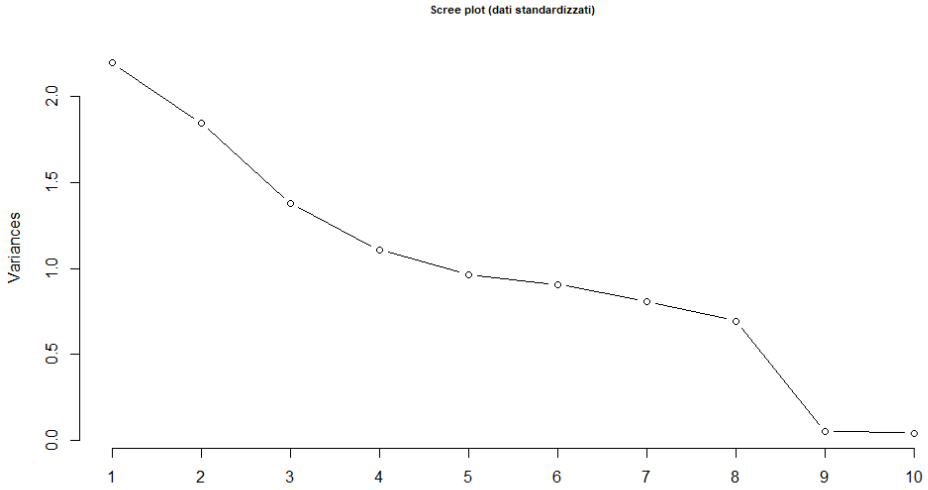


Figura 2.5: Scree plot dei dati standardizzati

```
> round(pc2$rotation,5)

##                          PC1       PC2       PC3
## display_text_width   0.09368  -0.25534   0.07320
## favorite_count       0.11839   0.02186  -0.68978
## retweet_count        0.10680   0.01187  -0.68758
## followers_count      0.64305   0.26381   0.09034
## friends_count        0.21993  -0.52202  -0.01211
## listed_count         0.62182   0.30601   0.12899
## statuses_count       0.30945  -0.49646   0.05619
## favourites_count     0.13573  -0.40860  -0.07751
## ripetizioni         -0.01496   0.02890   0.00740
## product_user_count   0.05354  -0.29015   0.10919

##                          PC4       PC5       PC6
## display_text_width   0.62469   0.10242  -0.22252
## favorite_count       0.08146  -0.01291   0.02686
## retweet_count        0.12636  -0.01155   0.02348
## followers_count      0.01226  -0.00364  -0.02251
## friends_count       -0.21860  -0.04207   0.45990
## listed_count         0.01684  -0.00237  -0.07404
## statuses_count      -0.10141  -0.01683   0.28114
## favourites_count    -0.39417   0.02613  -0.80122
```

```
## ripetizioni        -0.01632  -0.98495  -0.05809
## product_user_count 0.61075  -0.12781  -0.08571

##                            PC7           PC8
## display_text_width      0.68196       -0.01135
## favorite_count         -0.07287       -0.02644
## retweet_count           0.03207        0.06316
## followers_count        -0.00821       -0.04618
## friends_count           0.12534       -0.63258
## listed_count           -0.03238       -0.07543
## statuses_count         -0.03976        0.74819
## favourites_count       -0.06669       -0.07487
## ripetizioni             0.15522        0.02979
## product_user_count     -0.69401       -0.14464

##                            PC9           PC10
## display_text_width      0.07177       -0.00936
## favorite_count          0.70377       -0.02987
## retweet_count          -0.70277       -0.01149
## followers_count        -0.00085        0.71124
## friends_count          -0.03752       -0.02524
## listed_count           -0.02274       -0.70003
## statuses_count          0.04126       -0.04404
## favourites_count       -0.01853        0.01460
## ripetizioni             0.01206       -0.00099
## product_user_count     -0.03944        0.01400
```

Dalla matrice di rotazione si vede che la prima componente principale riflette le variabili `followers_count` (0.643) e `listed_count` (0.622), mentre la seconda `friends_count` (-0.522) e `statuses_count` (-0.496), con valore negativo. La terza componenente principale è rappresentata soprattutto da `favorite_count` (-0.690) e `retweet_count` (-0.687), la quarta da `display_text_width` (0.682) e `product_user_count` (-0.611), la quinta da `ripetizioni` (-0.985), la sesta da `favourites_count` (-0.801) e infine, la settima, positivamente da `display_text_width` (0.682) e negativamente da `product_user_count` (-0.694).

Per una rappresentazione grafica dei risultati ottenuti si può costruire il *biplot*. Viene riportato il grafico per le prime due componenti principali.

```
> #Biplot
> biplot(pc2, main="Biplot", xlim=c(-0.1,0.4),ylim
   =c(-0.25,0.15), cex=c(0.5,1))
```

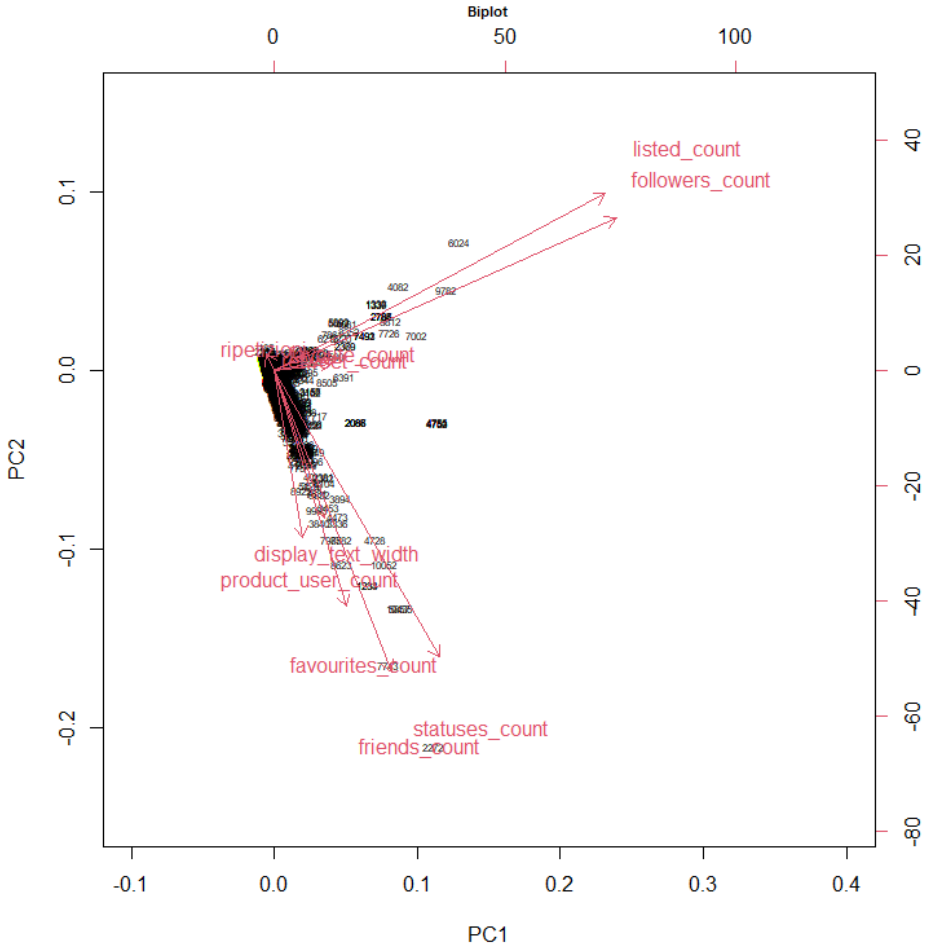


Figura 2.6: Biplot delle prime due componenti principali

Anche guardando il *biplot* si può affermare che le variabili `listed_count` e `followers_count` sono le maggiori rappresentati della prima componente principale e sono legate da una forte correlazione. La seconda componente principale è individuata soprattutto da `statuses_count` e `friends_count`. Le variabili `display_text_width`, `product_user_count`, `favourites_count`, `statuses_count` e `friends_count`

sono molto correlate tra di loro (e incorrelate con `listed_count` e `followers_count`).

Si trovano i nuovi valori delle osservazioni basate sulle 7 componenti principali che vengono tenute in considerazione.

```
> head(pc2$x)[,1:TwitterPCANumber]

##                   PC1          PC2           PC3
## [1,]      -0.68216374    1.2848600    0.07172013
## [2,]      -0.42554191    0.7865659   -0.05115957
## [3,]      -0.44653115    0.8248475   -0.03196178
## [4,]      -0.00797422   -0.2431174   -0.17800521
## [5,]      -0.20486523    0.1967760    0.14709136
## [6,]      -0.39972909    0.7280295   -0.01469685

##                          PC4               PC5
## [1,]              -1.0025608         17.8188736
## [2,]              -0.7046654          1.1351721
## [3,]              -0.7397388          2.2493257
## [4,]              -1.6085718          1.0975318
## [5,]               0.8963236          0.8538594
## [6,]              -0.4818737          1.0783696

##                          PC6               PC7
## [1,]              -0.6551228          2.0688917
## [2,]               0.3209588         -0.5362143
## [3,]               0.2600589         -0.3772213
## [4,]              -0.8287657         -0.5523356
## [5,]              -0.1856397          1.3344588
## [6,]               0.3014494         -0.1500227
```

I valori così trovati vengono aggiunti al dataframe originale.

```
> TwitterPCA <- (pc2$x[,1:TwitterPCANumber])
> for(i in 1:TwitterPCANumber) {
  colnames(TwitterPCA)[i] <- paste0("PC", i)
}

> colnames(TwitterPCA)
```

```
## [1] "PC1" "PC2" "PC3" "PC4" "PC5" "PC6" "PC7"

> TwitterPCA <- as.data.frame(TwitterPCA)
> Twitter.data <- cbind(Twitter.data, TwitterPCA)
> dim(Twitter.data)

## [1] 10456   104
```

Al dataframe `Twitter.data` sono state quindi aggiunte 7 nuove variabili, per un totale di 104 colonne.

2.4 Creazione del dataframe unico con i dati di Twitter e di Reddit

Una volta ottenuto il dataframe con le osservazioni filtrate e le nuove variabili, è il momento di normalizzare i dati in modo tale che possa essere creato un unico dataframe che contenga anche le osservazioni raccolte su Reddit. La fase di normalizzazione viene fatta ancora una volta su *RStudio*. Si passa poi su *Microsoft SQL Server Management Studio*, dove viene creata, su un server locale condiviso, una tabella che ha come colonne tutte le variabili in comune tra Twitter e di Reddit, più quelle eslusive di ciascun social network. Una volta creata, è possibile allocare all'interno i dati trovati in precedenza, tramite una connessione tra *RStudio* e il server.

2.4.1 Normalizzazione dei dati di Twitter

In vista dell'unione delle variabili di Twitter con quelle di Reddit, si decide di aggiungere prima di ogni variabile trovata la parola "Twitter". In questo modo, le variabili che risultano proprie di Twitter, vengono rese riconoscibili.

```
> #Aggiungo "Twitter" prima di ogni variabile
> n <- length(colnames(Twitter.data))
> for(i in 1:n) {
```

```
  colnames(Twitter.data)[i] <- paste0("Twitter_",
     colnames(Twitter.data)[i])
}
```

Per motivi di praticità, si decide di modificare ulteriormente i nomi delle variabili, eliminando il carattere *underscore* (_) e scrivendo i nomi in *camel case*, o *notazione a cammello*. Questo tipo di pratica consiste nello scrivere le parole tutte unite tra di loro, ma lasciando le iniziali maiuscole. L'utilità di presentare le variabili scritte con questa notazione riflette la possibilità, in *SQL*, di riferirsi alle variabili senza preoccuparsi della scrittura in minuscolo o in maiuscolo (infatti, in *SQL*, non vi è distinzione), ma al tempo stesso di presentarle in modo ordinato.

```
> #Trasformo tutte le variabili in CamelCase
> camel <- function(x){
  capit <- function(x) paste0(toupper(substring(x,
      1, 1)), substring(x, 2, nchar(x)))
  sapply(strsplit(x, "\\_"), function(x) paste(
     capit(x), collapse=""))
}
> colnames(Twitter.data) <- camel(colnames(Twitter
   .data))
```

Un'ulteriore modifica da attuare sui nomi delle variabili viene fatta su quelle che si riferiscono ai link presenti nei tweet sotto il dominio *t.co*, che è il modo con cui Twitter riduce URL lunghi. Viene rimosso il punto ., in quanto crea problemi di lettura in *SQL*.

```
> #Tolgo il "." nel nome di alcune variabili
> camel2 <- function(x){
  capit <- function(x) paste0(toupper(substring(x,
      1, 1)), substring(x, 2, nchar(x)))
  sapply(strsplit(x, "\\."), function(x) paste(
     capit(x), collapse=""))
}
> colnames(Twitter.data) <- camel2(colnames(
   Twitter.data))
```

Una volta modificate tutte le variabili relative a Twitter, è necessario

paragonare le variabili ottenute dalla ricerca di dati su Twitter e Reddit e scegliere un nome univoco per le variabili in comune. Le variabili che vengono modificate sono:

- `screen_name` diventa ***ContentUsername***. Indica l'*username* dell'utente autore del tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `text` diventa ***ContentText***. Indica il contenuto del tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `reply_to_screen_name` diventa ***ReplyToUsername***. Indica, nel caso di *tweet risposta*, l'username dell'utente a cui viene risposto, oppure nel caso di commento (Reddit), l'utente autore del post iniziale che viene commentato;
- `lang` diventa ***Lang***. Indica la lingua in cui viene scritto il tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `status_url` diventa ***ContentUrl***. Indica il link del tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `content_date` diventa ***ContentDate***. Indica la data in cui è stato scritto il tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `account_date` diventa ***AccountDate***. Indica la data in cui è stato creato l'account autore del tweet (quando si riferisce a Twitter) e del post o del commento (quando si riferisce a Reddit);
- `type_text` diventa ***TypeText***. Indica il tipo di contenuto a cui fa riferimento: nel caso di Twitter può specificare, come si è già visto, se il tweet è un *tweet risposta*, un *tweet citazione*, un retweet o un tweet indipendente; nel caso di Reddit, specifica se il contenuto è un post o un commento;
- `product_user_count` diventa ***ProductUserCount***. Indica quante volte l'utente autore del tweet (quando si riferisce a Twitter) e del

post o del commento (quando si riferisce a Reddit) sono presenti nel dataset;

- `social_media` diventa ***SocialMedia***. Indica il social network su cui è stata raccolta l'osservazione a cui fa riferimento.

```
> #Cambio il nome delle variabili in comune con
   altri social
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterScreenName"] <- "ContentUsername"
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterText"] <- "ContentText"
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterReplyToScreenName"] <- "ReplyToUsername"
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterLang"] <- "Lang"
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterStatusUrl"] <- "ContentUrl"
> colnames(Twitter.data)[colnames(Twitter.data)=="
   TwitterStatusesCount"] <- "ContentUserCount"
> colnames(Twitter.dataFILTRATO)[colnames(Twitter.
   dataFILTRATO)=="TwitterContentDate"] <- "
   ContentDate"
> colnames(Twitter.dataFILTRATO)[colnames(Twitter.
   dataFILTRATO)=="TwitterAccountDate"] <- "
   AccountDate"
> colnames(Twitter.dataFILTRATO)[colnames(Twitter.
   dataFILTRATO)=="TwitterTypeText"] <- "TypeText"
> colnames(Twitter.dataFILTRATO)[colnames(Twitter.
   dataFILTRATO)=="TwitterProductUserCount"] <- "
   ProductUserCount"
> colnames(Twitter.dataFILTRATO)[colnames(Twitter.
   dataFILTRATO)=="TwitterSocialMedia"] <- "
   SocialMedia"
```

Il nome delle variabili di Twitter da inserire nel database in *SQL* sono quindi:

```
> colnames(Twitter.data)

## [1]   "TwitterUserId" "TwitterStatusId"
```

```
## [3]   "ContentUsername" "ContentText"
## [5]   "TwitterSource" "TwitterDisplayTextWidth"
## [7]   "TwitterTwitterReplyToStatusId"
         "TwitterReplyToUserId"
## [9]   "ReplyToUsername" "TwitterIsQuote"
## [11]  "TwitterIsRetweet" "TwitterFavoriteCount"
## [13]  "TwitterRetweetCount" "TwitterQuoteCount"
## [15]  "TwitterReplyCount" "TwitterHashtags"
## [17]  "TwitterSymbols" "TwitterUrlsUrl"
## [19]  "TwitterUrlsTCo" "TwitterUrlsExpandedUrl"
## [21]  "TwitterMediaUrl" "TwitterMediaTCo"
## [23]  "TwitterMediaExpandedUrl"
         "TwitterMediaType"
## [25]  "TwitterExtMediaUrl" "TwitterExtMediaTCo"
## [27]  "TwitterExtMediaExpandedUrl"
         "TwitterExtMediaType"
## [29]  "TwitterMentionsUserId"
         "TwitterMentionsScreenName"
## [31]  "Lang" "TwitterQuotedStatusId"
## [33]  "TwitterQuotedText"
         "TwitterQuotedCreatedAt"
## [35]  "TwitterQuotedSource"
         "TwitterQuotedFavoriteCount"
## [37]  "TwitterQuotedRetweetCount"
         "TwitterQuotedUserId"
## [39]  "TwitterQuotedScreenName"
         "TwitterQuotedName"
## [41]  "TwitterQuotedFollowersCount"
         "TwitterQuotedFriendsCount"
## [43]  "TwitterQuotedStatusesCount"
         "TwitterQuotedLocation"
## [45]  "TwitterQuotedDescription"
         "TwitterQuotedVerified"
## [47]  "TwitterRetweetStatusId"
         "TwitterRetweetText"
## [49]  "TwitterRetweetCreatedAt"
         "TwitterRetweetSource"
## [51]  "TwitterRetweetFavoriteCount"
         "TwitterRetweetRetweetCount"
```

```
## [53]  "TwitterRetweetUserId"
        "TwitterRetweetScreenName"
## [55]  "TwitterRetweetName"
        "TwitterRetweetFollowersCount"
## [57]  "TwitterRetweetFriendsCount"
        "TwitterRetweetStatusesCount"
## [59]  "TwitterRetweetLocation"
        "TwitterRetweetDescription"
## [61]  "TwitterRetweetVerified"
        "TwitterPlaceUrl"
## [63]  "TwitterPlaceName" "TwitterPlaceFullName"
## [65]  "TwitterPlaceType" "TwitterCountry"
## [67]  "TwitterCountryCode" "TwitterGeoCoords"
## [69]  "TwitterCoordsCoords" "TwitterBboxCoords"
## [71]  "ContentUrl" "TwitterName"
## [73]  "TwitterLocation" "TwitterDescription"
## [75]  "TwitterUrl" "TwitterProtected"
## [77]  "TwitterFollowersCount"
        "TwitterFriendsCount"
## [79]  "TwitterListedCount" "ContentUserCount"
## [81]  "TwitterFavouritesCount"
        "TwitterVerified"
## [83]  "TwitterProfileUrl"
        "TwitterProfileExpandedUrl"
## [85]  "TwitterAccountLang"
        "TwitterProfileBannerUrl"
## [87]  "TwitterProfileBackgroundUrl"
        "TwitterProfileImageUrl"
## [89]  "ContentDate" "TwitterContentTime"
## [91]  "AccountDate" "TwitterAccountTime"
## [93]  "TwitterRipetizioni" "TwitterIsReply"
## [95]  "TypeText" "ProductUserCount"
## [97]  "SocialMedia" "TwitterPC1"
## [99]  "TwitterPC2" "TwitterPC3"
## [101] "TwitterPC4" "TwitterPC5"
## [103] "TwitterPC6" "TwitterPC7"
```

Con la normalizzazione dei dati si evita la ridondanza delle variabili che risulterebbero distinte ma che, di fatto, contengono le medesime

informazioni.

Prima di caricare i dati in *SQL*, è necessario modificare il formato di alcune variabili codificate come liste, trasformandole in stringhe.

Le liste diventano stringhe (per passaggio al database *SQL*).

```
> #Trasformo le liste in stringhe
> Twitter.data <- as.data.frame(Twitter.data)
> n <- dim(Twitter.data)[2]
> for(i in 1:n) {
  if(typeof(Twitter.data[,i])=="list") {
    Twitter.data[,i] <- paste(Twitter.data[,i])
  }
}
```

2.4.2 Caricamento dei dati in un database

Modificate i nomi e alcuni formati delle variabili su *RStudio*, i dati sono pronti per essere allocati sul database del server locale.

Si accede a *Microsoft SQL Server Management Studio*, che è un programma che offre gli strumenti per configurare, monitorare e amministrare le istanze di *SQL Server* e i database. [7] Inserendo i dati relativi al server locale (nel caso presente, si è utilizzato il server proprio dell'azienda *Omicron Consulting S.r.l*) con un dato nome utente e una password, è possibile creare un database chiamato `SocialMediaContent`. Un **database**, o una *base di dati*, è una collezione di dati persistenti organizzati e gestiti in modo efficace ed efficiente. Per *efficacia* ci si riferisce alla restituzione di tutte le informazioni di interesse di fronte alle richieste dell'utente, mentre l'*efficenza* consiste nello restituire le informazioni di interesse con il minor costo computazionale possibile. [8] Lavorare su una base di dati si è rivelato necessario soprattutto per due delle caratteristiche

[7] Documentazione Microsoft, *Scaricare SQL Server Management Studio (SSMS)*, `https://docs.microsoft.com/it-it/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15`

[8] Melucci M. Lezioni di Basi di Dati. Padova: C.L.E.U.P., 2020

proprie di questa tecnologia: in primo luogo, per garantire la *sicurezza* dei dati trovati, principalmente di fronte ad eventi "naturali" quali guasti o incidenti delle macchine (*safety*); in secondo luogo, per permettere la *condivisione dei dati* raccolti da macchine diverse in un'unica tabella condivisa.

All'interno del database `SocialMediaContent` si crea una tabella, chiamata `SocialMediaContentData`, formata dalle variabili uniche per Twitter, uniche per Reddit e dalle variabili in comune tra le due piattaforme, aggiungendo anche i dati relativi alle componenti principali (in questo caso, vengono inserite solo le colonne relative alle sette componenti principali di Twitter trovate, poiché non si sono compiute le analisi su Reddit. Si ribadisce che la procedura relativa alla *PCA* sarebbe stata la medesima anche per i dati di Reddit). Per costruire una tabella nel database, è necessario stabilire il nome della variabile e il tipo di dato che rappresenta.

Viene riportato il codice per la creazione della tabella `SocialMediaContentData`.

```
USE [AdvAnalytics]
GO

/****** Object:  Table [dbo].[
   SocialMediaContentData]    Script Date: 21/12/2
   021 13:42:02 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[SocialMediaContent](
[ContentUsername] [nvarchar](max) NULL,
[ContentText] [nvarchar](max) NULL,
[ReplyToUsername] [nvarchar](max) NULL,
[Lang] [nvarchar](50) NULL,
[ContentUrl] [nvarchar](max) NULL,
[ContentUserCount] [int] NULL,
[ProductUserCount] [int] NULL,
```

```
[ContentDate] [date] NULL,
[AccountDate] [date] NULL,
[TypeText] [nvarchar](50) NULL,
[SocialMedia] [nvarchar](max) NULL,
[TwitterUserId] [nvarchar](max) NULL,
[TwitterStatusId] [nvarchar](max) NULL,
[TwitterSource] [nvarchar](50) NULL,
[TwitterDisplayTextWidth] [int] NULL,
[TwitterReplyToStatusId] [nvarchar](max) NULL,
[TwitterReplyToUserId] [nvarchar](max) NULL,
[TwitterIsQuote] [bit] NULL,
[TwitterIsRetweet] [bit] NULL,
[TwitterFavoriteCount] [int] NULL,
[TwitterRetweetCount] [int] NULL,
[TwitterQuoteCount] [int] NULL,
[TwitterReplyCount] [int] NULL,
[TwitterHashtags] [nvarchar](max) NULL,
[TwitterSymbols] [nvarchar](max) NULL,
[TwitterUrlsUrl] [nvarchar](max) NULL,
[TwitterUrlsTCo] [nvarchar](max) NULL,
[TwitterUrlsExpandedUrl] [nvarchar](max) NULL,
[TwitterMediaUrl] [nvarchar](max) NULL,
[TwitterMediaTCo] [nvarchar](max) NULL,
[TwitterMediaExpandedUrl] [nvarchar](max) NULL,
[TwitterMediaType] [nvarchar](50) NULL,
[TwitterExtMediaUrl] [nvarchar](max) NULL,
[TwitterExtMediaTCo] [nvarchar](max) NULL,
[TwitterExtMediaExpandedUrl] [nvarchar](max) NULL,
[TwitterExtMediaType] [nvarchar](max) NULL,
[TwitterMentionsUserId] [nvarchar](max) NULL,
[TwitterMentionsScreenName] [nvarchar](max) NULL,
[TwitterQuotedStatusId] [nvarchar](max) NULL,
[TwitterQuotedText] [nvarchar](max) NULL,
[TwitterQuotedCreatedAt] [datetime] NULL,
[TwitterQuotedSource] [nvarchar](50) NULL,
[TwitterQuotedFavoriteCount] [int] NULL,
[TwitterQuotedRetweetCount] [int] NULL,
[TwitterQuotedUserId] [nvarchar](max) NULL,
[TwitterQuotedScreenName] [nvarchar](max) NULL,
```

```
[TwitterQuotedName] [nvarchar](max) NULL,
[TwitterQuotedFollowersCount] [int] NULL,
[TwitterQuotedFriendsCount] [int] NULL,
[TwitterQuotedStatusesCount] [int] NULL,
[TwitterQuotedLocation] [nvarchar](max) NULL,
[TwitterQuotedDescription] [nvarchar](max) NULL,
[TwitterQuotedVerified] [bit] NULL,
[TwitterRetweetStatusId] [nvarchar](max) NULL,
[TwitterRetweetText] [nvarchar](max) NULL,
[TwitterRetweetCreatedAt] [datetime] NULL,
[TwitterRetweetSource] [nvarchar](50) NULL,
[TwitterRetweetFavoriteCount] [int] NULL,
[TwitterRetweetRetweetCount] [int] NULL,
[TwitterRetweetUserId] [nvarchar](max) NULL,
[TwitterRetweetScreenName] [nvarchar](max) NULL,
[TwitterRetweetName] [nvarchar](max) NULL,
[TwitterRetweetFollowersCount] [int] NULL,
[TwitterRetweetFriendsCount] [int] NULL,
[TwitterRetweetStatusesCount] [int] NULL,
[TwitterRetweetLocation] [nvarchar](max) NULL,
[TwitterRetweetDescription] [nvarchar](max) NULL,
[TwitterRetweetVerified] [bit] NULL,
[TwitterPlaceUrl] [nvarchar](max) NULL,
[TwitterPlaceName] [nvarchar](max) NULL,
[TwitterPlaceFullName] [nvarchar](max) NULL,
[TwitterPlaceType] [nvarchar](50) NULL,
[TwitterCountry] [nvarchar](max) NULL,
[TwitterCountryCode] [nvarchar](50) NULL,
[TwitterGeoCoords] [nvarchar](max) NULL,
[TwitterCoordsCoords] [nvarchar](max) NULL,
[TwitterBboxCoords] [nvarchar](max) NULL,
[TwitterName] [nvarchar](max) NULL,
[TwitterLocation] [nvarchar](max) NULL,
[TwitterDescription] [nvarchar](max) NULL,
[TwitterUrl] [nvarchar](max) NULL,
[TwitterProtected] [bit] NULL,
[TwitterFollowersCount] [int] NULL,
[TwitterFriendsCount] [int] NULL,
[TwitterListedCount] [int] NULL,
```

```
[TwitterFavouritesCount] [int] NULL,
[TwitterVerified] [bit] NULL,
[TwitterProfileUrl] [nvarchar](max) NULL,
[TwitterProfileExpandedUrl] [nvarchar](max) NULL,
[TwitterAccountLang] [nvarchar](50) NULL,
[TwitterProfileBannerUrl] [nvarchar](max) NULL,
[TwitterProfileBackgroundUrl][nvarchar](max) NULL,
[TwitterProfileImageUrl] [nvarchar](max) NULL,
[TwitterContentTime] [time](7) NULL,
[TwitterAccountTime] [time](7) NULL,
[TwitterRipetizioni] [int] NULL,
[TwitterIsReply] [bit] NULL,
[RedditTitle] [nvarchar](max) NULL,
[RedditSubreddit] [nvarchar](max) NULL,
[RedditComments] [float] NULL,
[RedditProductTitle] [bit] NULL,
[RedditProductText] [bit] NULL,
[RedditCommentStart] [nvarchar](max) NULL,
[RedditCommentEnd] [nvarchar](max) NULL,
[RedditTopTokenComment] [nvarchar](max) NULL,
[RedditContentScore] [float] NULL,
[RedditSubredditsUser] [nvarchar](max) NULL,
[RedditTitlesUser] [nvarchar](max) NULL,
[RedditKarmaUser] [float] NULL,
[RedditSuspendedUser] [bit] NULL,
[RedditUserCountPost] [float] NULL,
[RedditUserCountComment] [float] NULL,
[TwitterPC1] [float] NULL,
[TwitterPC2] [float] NULL,
[TwitterPC3] [float] NULL,
[TwitterPC4] [float] NULL,
[TwitterPC5] [float] NULL,
[TwitterPC6] [float] NULL,
[TwitterPC7] [float] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Costruita in questo modo, si ha nel database una tabella con 104 colonne e 0 righe, in quanto non vi è stato inserito ancora alcun

dato.

Si deve quindi tornare su *RStudio* e stabilire una connessione con il server. Viene utilizzata la libreria `DBI`.

```
> #Caricamento libreria per effettuare la
   connessione
> library(DBI)
> #Connessione al server locale
> con1 <- DBI::dbConnect(odbc::odbc(),
                         Driver = ***,
                         Server = ***,
                         Database = "
                            SocialMediaContent",
                         Uid = ***,
                         Pwd = ***)
```

Creata la connessione con la base di dati, è possibile allocare i dati.

```
> #Allocazione dei dati nella tabella
> dbAppendTable(conn=con1, name="
   SocialMediaContent", value=Twitter.data)
```

La tabella presente ora sul database è formata da 10 456 righe, o *record*, formate a loro volta da 104 *campi* cui nome e tipo di dato sono stati definiti in precedenza. Nel caso presente, tutti i campi relativi ai dati su Reddit hanno, ovviamente, valore `NULL`.

2.5 Analisi testuale

Con l'analisi delle relazioni tra le variabili e la *PCA*, sono state prese in considerazione le variabili numeriche presenti nel dataset. Viene ora cambiato oggetto di interesse, e si sposta l'attenzione verso il contenuto testuale dei dati ottenuti dai social media considerati. Nel caso presente risultano infatti essere questi i dati più significativi e interessanti, nonché quelli che possono portare al raggiungimento dell'obiettivo iniziale di cogliere e interpretare l'opinione degli utenti riguardo al *product* di riferimento. Le variabili che vengono considerate non sono più dunque quelle numeriche, bensì `ContentText` e

le componenti principali trovate dalla *PCA*, che si dimostrano un ulteriore strumento utile alla spiegazione dei risultati. Dato che da ora in avanti nella trattazione si fa riferimento al contenuto testuale di ogni tweet, post o commento, ogni osservazione viene definita come "documento".

Tutte le fasi della procedura che seguono sono state effettuate su *Jupyter Notebook*, un'applicazione web che supporta il linguaggio di programmazione *Python*. *Python* mette a disposizione numerose nuove funzionalità rispetto ad *R*, soprattutto per quanto riguarda l'analisi del linguaggio naturale (*NLP*), con alcune librerie specifiche. Il suo utilizzo risulta quindi preferibile nel caso presente, oltre che ad essere un linguaggio di programmazione semplice da utilizzare, versatile e molto popolare.

Vengono riportati codice e risultati solo dei dati raccolti su Twitter, sebbene si ribadisce che la procedura risulti uguale aggiungendo anche i dati provenienti da Reddit.

2.5.1 Caricamento dei dati in Jupyter Notebook

Per trasferire tutti i dati della tabella `SocialMediaContent` sulla nuova piattaforma, viene utilizzata la libreria `pyodbc` che stabilisca una connessione tra il server locale e *Jupyter Notebook*.

```
import pyodbc

con = pyodbc.connect('DRIVER=***;'
                     'SERVER=***;'
                     'DATABASE=SocialMediaContent
                        ;'
                     'UID=***;'
                     'PWD=***;')
```

Viene importata la libreria `pandas`, a cui viene fatto riferimento con l'abbreviazione `pd`. `pandas` è una delle principali librerie di *Python* e contiene numerose funzioni per la gestione dei dataframe. La funzione utilizzata per la restituzione dei dati presenti nella

tabella del server locale è `read_sql_query`, a cui viene passata come parametro l'istruzione scritta in linguaggio *SQL* corrispondente.

```
import pandas as pd

DataFrame= pd.read_sql_query('SELECT␣*␣FROM␣
   SocialMediaContentData', con)

len(DataFrame)
## Out: 10456
```

Caricati i dati in *Jupyter Notebook*, si può iniziare a lavorare con i documenti.

2.5.2 Passaggi preliminari al clustering

Un modo per studiare e analizzare i documenti a disposizione consiste nel dividerli, secondo criteri di similarità, in gruppi, detti *cluster*. Per fare questo, è necessario applicare delle procedure a priori che permettano, prima di tutto, di escludere gli elementi superflui che potrebbero portare a conclusioni fuorvianti o errate (*data cleaning*) e di semplificare il più possibile gli elementi a disposizione (*lemmatizzazione*). In seguito, le informazioni vengono filtrate (*filtraggio dei token*) in modo da conservare solo gli elementi più interessanti e importanti. Vengono poi attuate delle procedure specifiche (*metodo tf-idf*) per garantire la buona riuscita del *clustering*.

Data cleaning e lemmatizzazione

Per lavorare con i dati testuali, viene utilizzata la libreria `nltk`. Il nome `nltk` sta per *Natural Language Toolkit* ed è una libreria che mette a disposizione diverse interfacce per manipolare, con strumenti di programmazione, le lingue scritte. [9]

```
#Caricamento libreria
import nltk
```

[9] NLTK Documentation, `https://www.nltk.org/`

Uno dei principali passaggi nell'analisi dei dati testuali riguarda il concetto di *tokenizzazione*. La tokenizzazione consiste nel dividere una porzione di testo, una frase, un'espressione, un paragrafo o un intero documento, in piccole unità dette *token*. I token possono essere parole, numeri oppure segni di punteggiatura. La tokenizzazione è essenziale quando si svolge un'analisi testuale poiché permette di identificare le parole che costituiscono la frase, e in questo modo riuscire ad interpretarne il significato [10]. La funzione che viene utilizzata nel codice che segue per attuare la tokenizzazione è `word_tokenize`, del pacchetto `tokenize` di `nltk`.

Come si è visto in precedenza, nei *tweet risposta* è sempre presente il nome dell'utente a cui ci si rivolge. Poiché al fine dell'analisi corrente questa informazione non è particolarmente rilevante, si decide di eliminare il nome utente dal contenuto testuale dei documenti nel caso di *tweet risposta*. Non vengono eliminate invece le menzioni presenti in tweet organici: infatti, se in diversi tweet ci si riferisce a certo utente, questo potrebbe essere direttamente collegato al prodotto di interesse, rendendo il nome utente un elemento significativo. Si va a ricercare quindi l'*username* menzionato nei *tweet risposta* tramite la tokenizzazione e lo si elimina, trasformando poi i token in una nuova stringa di testo "pulita".

```
for i in range(len(DataFrame)):
    if DataFrame.TwitterIsReply[i] == True:
        l=[]
        lnew = ""
        l = nltk.tokenize.word_tokenize(DataFrame.
            ContentText[i])
        for token in l:
            if token != DataFrame.ReplyToUsername[
                i]:
                lnew = lnew + "␣" + token
        DataFrame.ContentText[i]=lnew
```

[10] Analytics Vidha, *How to Get Started with NLP – 6 Unique Methods to Perform Tokenization*, https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/

Si lavora ora con la variabile `ContentText` per tutti i documenti. Vengono costruite due liste, `l1` dove sono inseriti gli elementi nulli restituiti da `ContentText`, `l2` dove sono invece inseriti tutti gli altri.

```
l1 = []
l2=[]

for i in range(len(DataFrame)):
    if DataFrame.ContentText[i] == None or
       DataFrame.ContentText[i] == "":
        Token.append(l1)
        a=1
    else:
        l2.append(DataFrame.ContentText[i])
```

I successivi passaggi riguardano diverse fasi del *data cleaning*:

- Vengono rimossi i link;

```
import re

#Rimozione dei link
link_regex = re.compile('((https?):((//)|(\\\\))
   +([\w\d:#@%/;$()~_?\+-=\\\.&](#!)?)*)', re.
   DOTALL)
for i in range(len(l2)):
    links = re.findall(link_regex,l2[i])
    for link in links:
        l2[i] = l2[i].replace(link[0], '')
```

- Vengono rimossi i simboli, tra cui i segni di punteggiatura e quelli rappresentanti le *emoji*;

```
import numpy as np

#Rimozione dei simboli
symbols = "'!\"#$%&()@*+-.,/:;<=>?[\]^_`{|}~\
                                        n
   "
for i in range(len(l2)):
    for j in symbols:
        l2[i] = np.char.replace(l2[i], j, '␣')
```

- Si uniformano tutte le lettere in carattere minuscolo;

```
for i in range(len(l2)):
    l2[i] = str(l2[i])
    l2[i] = nltk.tokenize.word_tokenize(l2[i])
    if l2[i] == []:
        l2[i] = [""]

#Minuscolo
for i in range(len(l2)):
    l2[i] = np.char.lower(l2[i])
```

- Si rimuovono le *stopwords*, cioè parole utilizzate molto di frequente all'interno del discorso (per esempio articoli, preposizioni, congiunzioni) che però qualitativamente non portano informazione.

```
#Rimozione delle stopword
from nltk.corpus import stopwords
stops = set(stopwords.words('english'))

l2Filtered=[]
for i in range(len(l2)):
    l222=[]
    for w in l2[i]:
        if (w not in stops):
            l222.append(w)
    l2Filtered.append(l222)
    if l2Filtered[i] == []:
        l2Filtered[i] = [""]
```

I documenti vengono nuovamente trasformati in stringhe (unendo quindi i token rimanenti dal *data cleaning*).

```
#Costruzione stringhe
l2FilteredStr=[]
for i in range(len(l2Filtered)):
    l222 = ""
    for w in l2Filtered[i]:
        l222 = l222 + "␣" + w
    l2FilteredStr.append(l222)
```

Un altro importante passaggio preliminare in vista del *clustering* consiste nella *lemmatizzazione*, cioè nella semplificazione e standardizzazione delle parole nella loro forma base. Per esempio, dopo la lemmatizzazione, i verbi nei diversi tempi e forme vengono trasformati all'infinito; i sostantivi e gli aggettivi plurali sono tradotti al singolare; gli avverbi diventano sostantivi o aggettivi. [11] Proprio per la diversità con cui le differenti parti del discorso "reagiscono" alla lemmatizzazione, viene introdotta la funzione `get_wordnet_pos`, che permette di riconoscere a quali categorie lessicali appartengono le parole a disposizione. In questo modo la procedura di lemmatizzazione è diversa per ogni token.

```
from nltk.corpus import wordnet

def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
    "N": wordnet.NOUN,
    "V": wordnet.VERB,
    "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

#Lemmatizzazione
l2FilteredNew=[]
for i in range(len(l2FilteredStr)):
    l222 = []
    for w in l2FilteredStr[i]:
        l222 = nltk.tokenize.word_tokenize(l2
           FilteredStr[i])
    l2FilteredNew.append(l222)

l2FilteredNewLemmatization = nltk.stem.
   WordNetLemmatizer()
l2LemmatizationEnglish=[]
```

[11] Manning C.D.; Raghavan P.; Schütze H. Introduction to Information Retrieval, *Stemming and lemmatization*, https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

```
for i in range(len(l2FilteredNew)):
    l222=[]
    for words in l2FilteredNew[i]:
        l222.append(l2FilteredNewLemmatization.
            lemmatize(words, get_wordnet_pos(words)
            ))
    l2LemmatizationEnglish.append(l222)
```

Nella lista `Token` vengono inseriti tutti i token "puliti" ottenuti con i passaggi precedenti, divisi per ogni documento. Viene anche costruito il "dizionario" `df`, che conta il numero dei documenti in cui un determinato token è presente.

```
#Creazione lista
Token=[]
for i in range(len(l2LemmatizationEnglish)):
    Token.append(l2LemmatizationEnglish[i])

from collections import defaultdict
df = defaultdict(int)

for i in range(len(Token)):
    for token in np.unique(Token[i]):
        df[token]+=1
```

Con le informazioni contenute in `Token` e in `df` si possono costruire ulteriori dataframe che permettono di riconoscere i token utilizzati più di frequente, che risultano anche quelli più interessanti dal punto di vista dell'analisi.

Filtraggio dei token

Viene creata un'ulteriore lista, `TotalToken`, che come `Token` contiene i token, ma con la differenza che questi sono presenti una sola volta e non sono più divisi per documento. La lista viene successivamente trasformata in un array.

```
TotalToken = []
for i in range(len(DataFrame)):
```

```
    TotalToken = TotalToken + Token[i]
TotalToken = np.unique(TotalToken)

TotalToken
## Out: array(['0', '00', '000', ..., '  \xad', '
     \xad  ', '  '], dtype='<U42')
```

`TotalFreq` è un dizionario che contiene la frequenza relativa alla coppia *(i, j)*, dove *i* è riferito al numero del documento e *j* è riferito ad un determinato token. `TotalFreq` contiene quindi l'informazione relativa a quante volte viene ripetuto il token j nel documento i-esimo.

```
from collections import Counter,defaultdict

TotalFreq = {}
for i in range(len(DataFrame)):
    CountToken = Counter(Token[i])
    for token in np.unique(Token[i]):
        TotalFreq[i,token] = CountToken[token]

TotalFreq
## Out: {(0, '13'): 1,
         (0, 'cutie'): 1,
         (0, 'iphone'): 1,
         ...}
```

I precedenti passaggi sono finalizzati alla costruzione del dataframe `TotalFreqDB`, che ha come colonne i token ottenuti e come righe i numeri dei documenti: le intersezioni *(i, j)* tra righe e colonne indicano quante volte, all'interno del documento *i*, il token *j* è presente.

```
TotalFreqDBProv = pd.DataFrame(list(TotalFreq.
   items()),columns = ['column1','column2'])
TotalFreqDB = np.zeros((len(DataFrame),len(
   TotalToken)))
TotalFreqDB = pd.DataFrame(TotalFreqDB,columns =
   list(TotalToken))

for i in range(len(TotalFreqDBProv)):
```

```
    TotalFreqDB[(TotalFreqDBProv.column1[i])[1]][(
        TotalFreqDBProv.column1[i])[0]] =
        TotalFreqDBProv.column2[i]

TotalFreqDB
## Out:
```

	0	00	000	0000	000sdg	002	003	005	007	008	...	i	well	□	□check	ä	é	ø	ø	øø	ù
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10451	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10452	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10453	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10454	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10455	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10456 rows × 14170 columns

Figura 2.7: Dataframe *TotalFreqDB*

Il dataframe `TotalFreqDB` ha 10 456 righe e 14 170 colonne.

Come detto in precedenza, è d'interesse valutare se tutti i 14 170 token ottenuti siano d'interesse per le analisi. Tra questi, infatti, possono nascondersi alcune parole scritte in modo errato, parole inventate o parole che non risultano pertinenti con il tema d'interesse e riportate solo da una minoranza di utenti. È bene analizzare le frequenze di ogni token nel dataframe completo e scegliere una soglia sopra la quale ritenere un token "interessante" e "informativo" per le successive analisi e il clustering.

Per sapere quante volte il token j è presente in totale, cioè considerando tutti i documenti insieme, si sommano i valori delle colonne del dataframe `TotalFreqDB`, ottenendo i valori `s`.

```
#Somma colonne
s= TotalFreqDB.sum(axis=0)

s
```

```
## Out: 0          76.0
        00          28.0
        000        148.0
        ...
                    30.0
Length: 14170, dtype: float64
```

Gli stessi risultati vengono riportati nel dataframe `DataFrameTotalFreq`, che ha nella colonna `Token` i diversi token, e nella colonna `TotalFreq` il valore di `s` corrispondente.

```
DataFrameTotalFreq = pd.DataFrame.transpose(pd.
   DataFrame([s.index, s], index=["Token", "
   TotalFreq"]))
DataFrameTotalFreq
## Out:
```

	Token	TotalFreq
0	0	76.0
1	00	28.0
2	000	148.0
3	0000	8.0
4	000sdg	1.0
...	...	...
14165	é	2.0
14166	ø	24.0
14167	ø	1.0
14168	øø	1.0
14169	ù	30.0

14170 rows × 2 columns

Figura 2.8: DataFrame *DataFrameTotalFreq*

Con le informazioni a disposizione, si è deciso di conservare tutti i token con un valore `TotalFreq` (del dataframe `DataFrameTotalFreq`) maggiore di 100. Per quanto riguarda i token che presentano un valore minore, si è invece deciso di mantenere come token i numeri

e le parole di senso compiuto riconosciute dalla funzione `synset` del pacchetto `wordnet` della libreria `nltk`. Facendo così, si eliminano con una buona probabilità i token errati e poco pertinenti.

```
from num2words import num2words
from decimal import InvalidOperation

for j in range(101):
    DataFrameTotalFreqPROVA = DataFrameTotalFreq[
       DataFrameTotalFreq.TotalFreq==j]
    DataFrameTotalFreqPROVA.index=range(len(
       DataFrameTotalFreqPROVA))
    for i in range(len(DataFrameTotalFreqPROVA)):
        if (wordnet.synsets(
           DataFrameTotalFreqPROVA.Token[i]) ==
           []):
            try:
                num2words(DataFrameTotalFreqPROVA.
                   Token[i])
                raise InvalidOperation ("Error␣
                   message")
            except InvalidOperation as e:
                DataFrameTotalFreq=
                   DataFrameTotalFreq.drop(
                   DataFrameTotalFreq[
                   DataFrameTotalFreq.Token==
                   DataFrameTotalFreqPROVA.Token[i
                   ]].index,axis=0)

DataFrameTotalFreq.index=range(len(
   DataFrameTotalFreq))
DataFrameTotalFreq
## Out:
```

	Token	TotalFreq
0	0	76.0
1	000	148.0
2	1	418.0
3	10	365.0
4	100	69.0
...	...	...
6108	zoom	34.0
6109	□	539.0
6110	□	1121.0
6111	□	627.0
6112		442.0

6113 rows × 2 columns

Figura 2.9: Dataframe *DataFrameTotalFreq* dopo l'eliminazione dei token errati e poco pertinenti

Il dataframe `DataFrameTotalFreq` contiene ora 6 113 token su cui concentrare le analisi successive.

Metodo tf-idf

L'algoritmo *tf-idf* viene utilizzato per quantificare l'importanza dei diversi token all'interno dei documenti. Il nome *tf-idf* è composto da due acronimi: *tf* che sta per **Term Frequency** (frequenza del termine) e *idf* che sta per **Inverse Document Frequency** (frequenza inversa del documento): entrambi questi valori vanno calcolati e combinati per la riuscita dell'algoritmo.

Dati N documenti di una base di dati, si prende in considerazione il termine K. Il metodo *tf-idf* assegna un "punteggio" minore al termine K se questo è comune a molti documenti, mentre il peso è maggiore se K è presente solo in poche risorse. Se K è un termine "raro", infatti, l'efficacia della selezione è più elevata.

Il valore di **idf** viene calcolato con la seguente formula:

$$IDF_K = \frac{log(N)}{n_K}$$

dove N è il numero dei documenti in totale e n_K il numero dei documenti in cui il termine K è presente. Se il valore di n_K si avvicina a quello di N, il valore di IDF è prossimo allo zero e questo consegue che i token più comuni (cioé quelli presenti in molti documenti) hanno un peso nullo.

tf è dato invece da

$$TF_{XK} = \frac{O_{KX}}{D_X}$$

con O_{KX} che indica l'occorrenza del termine K nel documento X e D_X il numero delle parole del documento X.

I valori di *tf* e *idf* trovati vengono poi moltiplicati nella formula di **tf-idf**

$$TF - IDF = TF_{XK} IDF_K$$

che è elevato quando il termine K è molto frequente in un piccolo insieme di documenti, ma assente in tutti gli altri. [12]

Il metodo *tf-idf* risulta molto utile per organizzare i documenti e facilitare dunque il *clustering*.

Viene creato il dataframe `DataFrameTfIdf`, che ha come colonne i diversi token j, come righe i documenti i e come intersezioni il valore di *tf-idf* per la coppia *(i, j)*. Si noti che nella riga del calcolo del valore di `idf`, il rapporto presenta come denominatore `(df+1)` e non semplicemente `df` perché quest'ultimo potebbe avere valore zero.

```
tf = {}
tf_idf = {}
for i in range(len(DataFrame)):
    NumToken = len(Token[i])
    CountToken = Counter(Token[i])
    for token in np.unique(Token[i]):
```

[12] OkPedia, *TF-IDF*, `https://www.okpedia.it/tf-idf`

```
        if token in list(DataFrameTotalFreq.Token)
           :
            tf[i,token] = CountToken[token] /
               NumToken
            idf = np.log(len(DataFrame)/(df[token
               ]+1))
            tf_idf[i, token] = tf[i,token]*idf

tf
## Out: {(0, '13'): 0.3333333333333333,
         (0, 'cutie'): 0.3333333333333333,
         (0, 'iphone'): 0.3333333333333333,
         ...}

tf_idf
## Out: {(0, '13'): 0.07927054033027678,
         (0, 'cutie'): 1.4147653204026007,
         (0, 'iphone'): 0.057853754874945075,
         ...}

df1 = pd.DataFrame(list(tf_idf.items()),columns =
   ['column1','column2'])
DataFrameTfIdf = np.zeros((len(DataFrame),len(
   DataFrameTotalFreq)))
DataFrameTfIdf = pd.DataFrame(DataFrameTfIdf,
   columns = list(DataFrameTotalFreq.Token))

for i in range(len(df1)):
    DataFrameTfIdf[(df1.column1[i])[1]][(df1.
       column1[i])[0]] = df1.column2[i]

DataFrameTfIdf
## Out:
```

	0	000	1	10	100	1000	10000	100th	101	10th	...	zero	zimbabwe	zip	zipper	zombie	zoom	□	□	□	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.719977	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10451	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
10452	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
10453	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.276039	0.0
10454	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.199361	0.0
10455	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0

10456 rows × 6113 columns

Figura 2.10: Dataframe *DataFrameTfIdf*

Normalizzazione per il clustering

Per attuare il *clustering* vengono utilizzati i valori precedentemente trovati dati dall'algoritmo *tf-itf* e dai valori delle componenti principali dati dalla *PCA*.

Per entrambi i gruppi di dati, per un'esecuzione ottimale del *clustering* è necessario attuare una normalizzazione che renda tutti i valori confrontabili su una scala che va da 0 a 1.

Per i valori *tf-idf* si decide di utilizzare la normalizzazione ℓ^2, cioè la normalizzazione euclidea.

I valori *tf-idf* normalizzati vengono inseriti nel nuovo dataframe `DataFrameTfIdfScaled`.

```
from sklearn.preprocessing import Normalizer

DataNormalizer = Normalizer(norm='l2').fit(
   DataFrameTfIdf)
DataNormalized = DataNormalizer.transform(
   DataFrameTfIdf)
DataFrameTfIdfScaled = pd.DataFrame(DataNormalized
   ,columns = list(DataFrameTotalFreq.Token))

DataFrameTfIdfScaled

## Out:
```

	0	000	1	10	100	1000	10000	100th	101	10th	...	zero	zimbabwe	zip	zipper	zombie	zoom	□	□	□	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.560558	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10451	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
10452	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
10453	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.166397	0.0
10454	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.169221	0.0
10455	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0

10456 rows × 6113 columns

Figura 2.11: Dataframe *DataFrameTfIdfScaled*

Per quanto riguarda le componenti principali, si procede con la standardizzazione *Min-Max*. I valori delle componenti principali standardizzati si inseriscono all'interno del dataframe `ScaledPCA`.

```
from sklearn.preprocessing import MinMaxScaler

#Dataframe con solo le componenti principali.
DataFramePCA=DataFrame[["TwitterPC1", "TwitterPC2"
   ,"TwitterPC3","TwitterPC4","TwitterPC5","
   TwitterPC6","TwitterPC7"]]

scaler = MinMaxScaler()
ScaledPCA = scaler.fit_transform(DataFramePCA)
ScaledPCA=pd.DataFrame(ScaledPCA,columns=['
   TwitterPC1','TwitterPC2','TwitterPC3','
   TwitterPC4','TwitterPC5','TwitterPC6','
   TwitterPC7'])

ScaledPCA
## Out:
```

	TwitterPC1	TwitterPC2	TwitterPC3	TwitterPC4	TwitterPC5	TwitterPC6	TwitterPC7
0	0.005030	0.361299	0.802571	0.342476	0.236391	0.428965	0.545232
1	0.007289	0.354785	0.801325	0.355077	0.022354	0.454676	0.354233
2	0.007104	0.355286	0.801520	0.353593	0.036648	0.453072	0.365890
3	0.010963	0.341325	0.800038	0.316841	0.021872	0.424392	0.353051
4	0.007569	0.353721	0.801930	0.373145	0.021214	0.451980	0.396350
...	...	...	...	...	...	...	...
10451	0.014852	0.325143	0.800876	0.371373	0.003132	0.373139	0.472104
10452	0.007960	0.352548	0.801815	0.372486	0.006948	0.454746	0.386087
10453	0.014850	0.328601	0.800526	0.332305	0.007130	0.441861	0.405010
10454	0.011791	0.338316	0.801319	0.365630	0.005362	0.417679	0.419213
10455	0.007871	0.352799	0.801762	0.371765	0.006884	0.452010	0.384192

10456 rows × 7 columns

Figura 2.12: Dataframe *ScaledPCA*

I dataframe `DataFrameTfIdfScaled` e `ScaledPCA` vengono uniti nel dataframe `DataFrameScaled`, che contiene dunque i valori *tf-idf* e i valori delle componenti principali standardizzati.

```
DataFrameScaled=DataFrameTfIdfScaled.join(
   ScaledPCA)
```

È con i dati presenti in questo dataframe che viene svolto il *clustering*.

2.5.3 Clustering

Il **clustering**, o *raggruppamento*, è un metodo "non supervisionato" con cui la macchina organizza un insieme di dati in gruppi, o *cluster*, in base ad un criterio di somiglianza, distanza o vicinanza.

I dati sono visti come punti di uno spazio vettoriale reale di n dimensioni; d è una funzione $d(x,y)$ di distanza che associa a (x,y) un valore reale non negativo. Un **cluster** è rappresentato da un insieme di punti per cui ogni coppia di essi ha una distanza minore di una determinata soglia. All'interno di un cluster vi sono quindi elementi che presentano tra loro delle similarità, e delle dissimilarità con elementi di altri cluster. L'input di un algoritmo di *clustering* è

individuato da un insieme di dati e da un numero intero k, mentre l'output è l'insieme dei k cluster nei quali punti sono più vicini tra essi di quanto lo siano rispetto ai punti di tutti gli altri cluster. [13]

Nel caso presente, i dati che è d'interesse raggruppare in cluster sono i documenti.

Esistono diversi algoritmi di *clustering*. Uno di quelli maggioremente diffusi è il **K-means**, o delle *k medie*. Si tratta di un metodo *non gerarchico* in cui è necessario scegliere a priori il numero di *cluster* da costruire, ma i risultati possono essere confrontati per diverse numerosità. Per valutare il numero adatto di cluster possono essere utilizzati due indici: la somma dei quadrati dei residui (**SSE**) e la **Silhouette**. Si utilizzano le funzioni presenti nella libreria `sklearn`.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kmeans_kwargs = {
             "init": "random",
             "n_init": 10,
             "max_iter": 300,
             "random_state": 42}

sse = []
SilhouetteCoefficients = []
for k in range(1, 21):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(DataFrameScaled)
    sse.append(kmeans.inertia_)
    if k>1:
        score = silhouette_score(DataFrameScaled,
           kmeans.labels_)
        SilhouetteCoefficients.append(score)
```

L'indice *SSE* è dato dalla somma delle distanze al quadrato tra i centroidi (cioè il punto medio) e ogni elemento del cluster. Si

[13] Melucci M. *Lezioni di Basi di Dati*. Padova: C.L.E.U.P., 2020

sceglie un *range* di valori K, per esempio da 1 a 20, che corrisponde al diverso numero dei possibili cluster, e si calcola il valore della somma dei quadrati dei residui per ogni caso. I valori così ottenuti vengono inseriti all'interno di un grafico per determinare il "punto di gomito" (chiamato nel codice "ginocchio"), cioè quel valore K in cui la curva subisce un'inflessione [14]. Il K a cui corrisponde il punto di gomito è il numero ideale di cluster da costruire.

```
import matplotlib.pyplot as plt

plt.style.use("fivethirtyeight")
plt.plot(range(1,21), sse)
plt.xticks(range(1,21))
plt.xlabel("Number␣of␣Clusters")
plt.ylabel("SSE")
plt.show()
## Out:
```

Figura 2.13: Grafico che rappresenta il valore dell'indice *SSE* al variare del numero dei cluster

```
from kneed import KneeLocator
```

[14] Bonaros B. Predictive Hacks, *K-Means Elbow Method Code For Python*, `https://predictivehacks.com/k-means-elbow-method-code-for-python/`

```
kl = KneeLocator(
          range(1, 21), sse, curve="convex",
             direction="decreasing"
      )
kl.elbow
knee=kl.elbow
knee

## Out: 6
```

In questo caso, il programma riconosce come punto di gomito il valore $K = 6$.

Un secondo indice che può essere utilizzato per l'identificazione del numero di cluster è l'indice di *Silhouette*, o *indice di bontà della classificazione*. Viene calcolato come:

$$S(x*) = \frac{d_0 - d(x*, G_j)}{max\{d_0, d(x*, G_j)\}}$$

con x^* generica osservazione, G_j il gruppo al quale x^* appartiene e d_0 la distanza di x^* dal gruppo più vicino e diverso da quello cui appartiene. [15] Si ricerca il valore K tale per cui la *Silhouette* presenta un valore elevato, tenendo presente che non può mai presentare un valore maggiore di 1.

```
plt.style.use("fivethirtyeight")
plt.plot(range(2, 21), SilhouetteCoefficients)
plt.xticks(range(2, 21))
plt.xlabel("Number␣of␣Clusters")
plt.ylabel("Silhouette␣Coefficient")
plt.show()
## Out:
```

[15] Dispense del corso *Analisi dei Dati Multidimensionali*

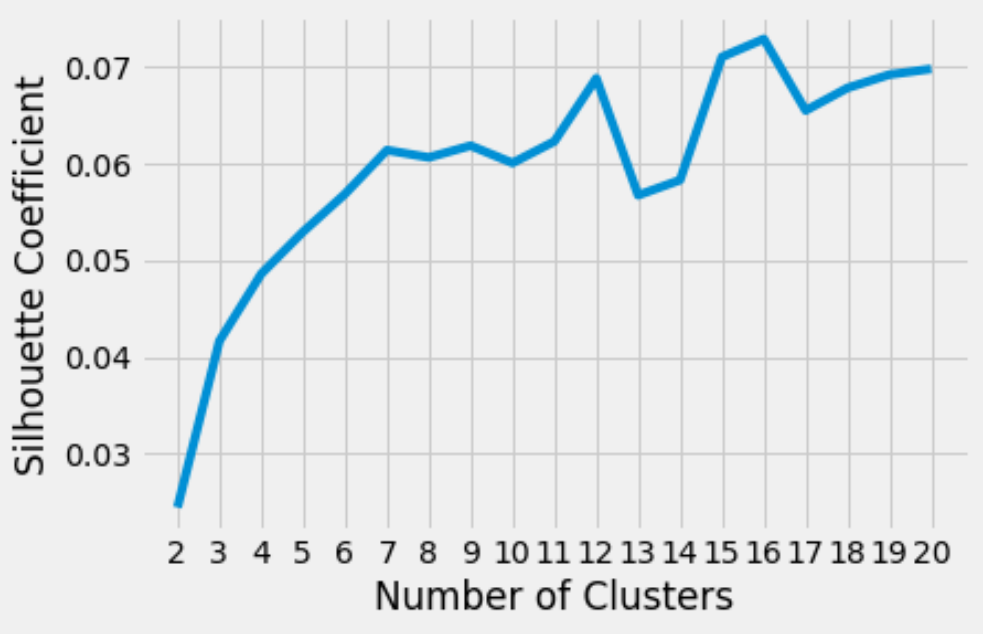


Figura 2.14: Grafico che rappresenta il valore dell'indice *Silhouette* al variare del numero dei cluster

Anche secondo questo indice, il valore 6 risulta essere un buon numero di cluster.

Si noti che l'indice di *Silhouette* è ben lontano da avere valore 1, come anche i valori di *SSE* risultano essere molto elevati.

Si sceglie dunque $K = 6$.

Per rappresentare i cluster graficamente, è necessario trasformare i dati in due dimensioni in funzione delle prime due componenti principali.

```
from sklearn.decomposition import PCA

pca = PCA(2)
pca.fit_transform(DataFrameScaled)
## Out: array([[-0.0693679 , -0.02861273],
       [-0.06292533, -0.04250071],
       [-0.0641647 , -0.01001289],
       ...,
       [-0.0128562 , -0.07518661],
       [-0.01170183, -0.08674398],
       [ 0.00881583, -0.05832453]])
```

```
#Trasformazione dei dati
DataFrameScaledPCA = pd.DataFrame(pca.fit_
   transform(DataFrameScaled),columns=['Component1
   ','Component2'])
DataFrameScaledPCA

## Out:
```

	Component1	Component2
0	-0.069366	-0.028609
1	-0.062924	-0.042496
2	-0.064164	-0.010009
3	-0.606327	0.727660
4	-0.010945	-0.031497
...	...	...
10451	-0.004273	-0.081931
10452	0.003653	-0.052608
10453	-0.012856	-0.075187
10454	-0.011702	-0.086744
10455	0.008816	-0.058325

10456 rows × 2 columns

Figura 2.15: Dataframe *DataFrameScaledPCA*

Si rappresentano graficamente i 6 cluster in cui sono stati divisi i 10 456 documenti.

```
import seaborn as sns

kmeans.fit(DataFrameScaledPCA)
PredictedLabels = kmeans.labels_
DataFrameScaledPCA["Cluster"] = kmeans.labels_
DataFrameScaled["Cluster"] = kmeans.labels_

plt.style.use("fivethirtyeight")
```

```
plt.figure(figsize=(8, 8))
scat = sns.scatterplot("Component1", "Component2",
    s=100, data=DataFrameScaledPCA, hue="Cluster",
    style="Cluster", palette="Set2")
scat.set_title("Clustering")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2,
   borderaxespad=0.0)
plt.show()
```

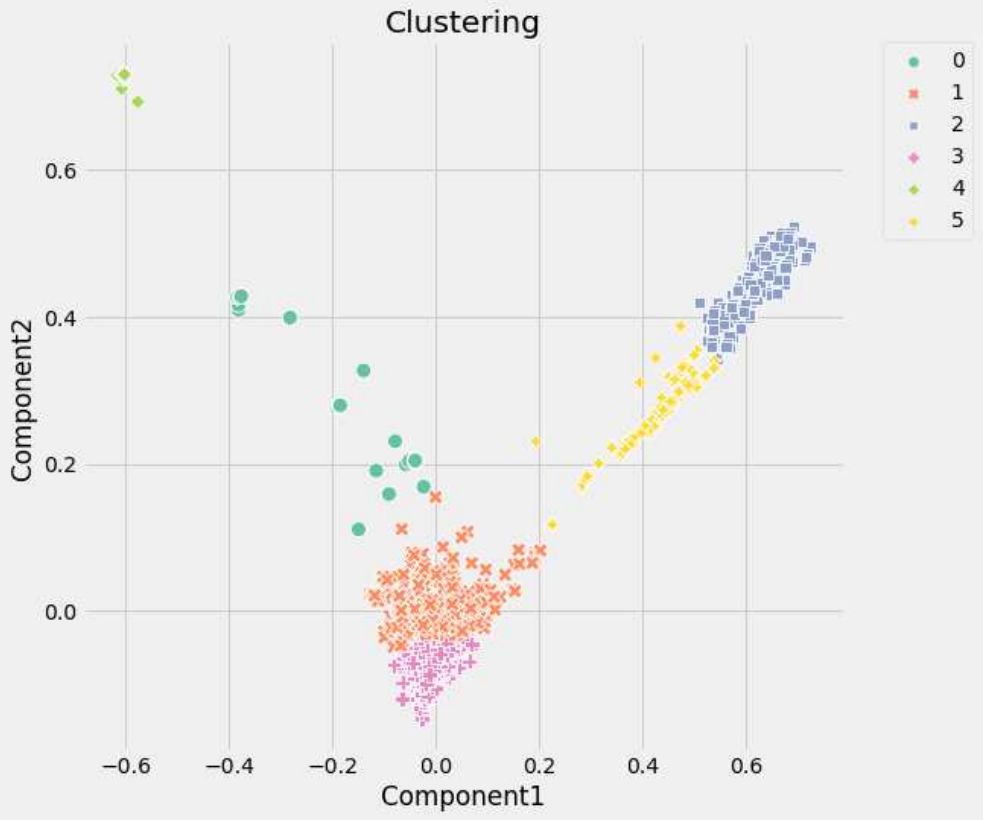


Figura 2.16: Rappresentazione grafica della distribuzione dei cluster secondo le prime due componenti principali

Il grafico è un ulteriore strumento che permette di stabilire l'efficacia del *clustering* svolto. Nello spazio i gruppi costruiti sono ben differenziati tra di loro, senza particolari sovrapposizioni tra gli elementi.

2.5.4 Caratterizzazione dei cluster

Una volta individuati i cluster, è doverso andare ad esplorarli e a caratterizzarli. Questo si può fare in un primo momento individuando i token maggiormente presenti all'interno dei documenti di ciascun cluster, successivamente stabilendo alcune *regole di associazione* tra di essi.

Individuazione dei token più frequenti

Per verificare quali sono i token maggiormente presenti all'interno di ogni cluster, è necessario recuperare i dataframe `TotalFreqDB` (che contiene le frequenze per cui il token j era presente nel documento i) e `DataFrameTfIdf` (che contiene i valori trovati con il metodo *tf-idf* per ogni token j e documento i). Si tengono in considerazione le frequenze assolute contenute in `TotalFreqDB` per i soli token presenti in `DataFrameTfIdf`, cioè quelli ottenuti dopo il filtraggio: per fare questo si cercano le colonne in comune tra i due dataframe.

```
CommonColumns= np.intersect1d(TotalFreqDB.columns,
    DataFrameTfIdf.columns)
TotalFreqDB=TotalFreqDB[list(CommonColumns)]
TotalFreqDB["Cluster"] = kmeans.labels_

TotalFreqDB["Cluster"]
## Out: 0         1
        1         1
        2         1
        3         4
        4         1
                 ..
        10451     3
        10452     3
        10453     3
        10454     3
        10455     3
Name: Cluster, Length: 10456, dtype: int32
```

Al dataframe `TotalFreqDB` è stata aggiunta la colonna `Cluster`, che indica a quale cluster appartiene il documento i a cui si riferisce.

Il codice seguente produce come output i primi 10 token più frequenti ogni per cluster.

```
TokenFreq=[]
for i in range(knee):
    S1=[]
    S1= TotalFreqDB1[TotalFreqDB1.Cluster==i].drop
       ('Cluster',axis=1).sum(axis=0) DataFrameS1
       = pd.DataFrame.transpose(pd.DataFrame([S1.
       index, S1], index=["Token", "TotalFreq"]))
    SortS1 = DataFrameS1.sort_values("TotalFreq",
       axis = 0, ascending = False)
    TokenFreq.append(SortS1[:10])

TokenFreq
## Out: [             Token  TotalFreq
          2901        iphone      128.0
          18              13      110.0
          4230           pro      107.0
          3372           max       62.0
          2361           get        1.0

                      Token  TotalFreq
          2901        iphone     4211.0
          18              13     3969.0
          4230           pro     2499.0
          3372           max     1597.0
          2361           get      353.0
          954           case      350.0
          3665           new      316.0
          392          apple      280.0
          4036         phone      279.0
          903         camera      240.0,

                      Token  TotalFreq
          1705         dome     1058.0
```

```
      2901       iphone     1024.0
      18             13     1016.0
      2388        glass     1009.0
      4284    protector      737.0
      328        amazon      687.0
      4230          pro      606.0
      2030           ez      573.0
      3372          max      565.0
      4767       screen      433.0,

             Token  TotalFreq
      2901       iphone     5265.0
      18             13     3934.0
      4230          pro     2019.0
      4036        phone     1336.0
      2361          get     1273.0
      392         apple     1142.0
      3665          new     1139.0
      2902     iphone13     1025.0
      6110            -     1018.0
      3372          max      844.0,

             Token  TotalFreq
      2901       iphone      316.0
      18             13      315.0

             Token  TotalFreq
      1705        dome      133.0
      18             13      125.0
      2901       iphone      124.0
      5995   whitestone      121.0
      954          case      113.0
      328        amazon      105.0
      1693       docomo       89.0
      4282   protection       78.0
      4230          pro       76.0
      4001      perfect       74.0]
```

Per rappresentare graficamente i risultati ottenuti, si utilizzano le *wordcloud*, ovvero rappresentazioni grafiche della frequenza delle parole, dove le parole più frequenti sono visualizzate in una grandezza maggiore. In questo caso, le *wordcloud* risultanti contengono tutti i token presenti all'interno dei cluster.

```
from wordcloud import WordCloud

TokenFreq=[]
for i in range(knee):
    S1=[]
    S1= TotalFreqDB1[TotalFreqDB1.Cluster==i].drop
       ('Cluster',axis=1).sum(axis=0) #somma
       colonne
    DataFrameS1 = pd.DataFrame.transpose(pd.
       DataFrame([S1.index, S1], index=["Token", "
       TotalFreq"]))
    SortS1 = DataFrameS1.sort_values("TotalFreq",
       axis = 0, ascending = False)
    TokenFreq.append(SortS1)

for i in range(knee):
    TokenFreqProv = []
    TokenFreqDict = defaultdict(int)
    TokenFreqProv = TokenFreq[i]
    TokenFreqDict = TokenFreqProv.set_index('Token
       ').to_dict()['TotalFreq']
    wc = WordCloud(width=800, height=400).generate
       _from_frequencies(TokenFreqDict)
    plt.figure(figsize=(10, 10))
    plt.title('Wordcloud␣per␣il␣cluster␣'+ str(i),
        fontsize=25)
    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")
    plt.figure()
```

Wordcloud per il cluster 0

Wordcloud per il cluster 1

Wordcloud per il cluster 2

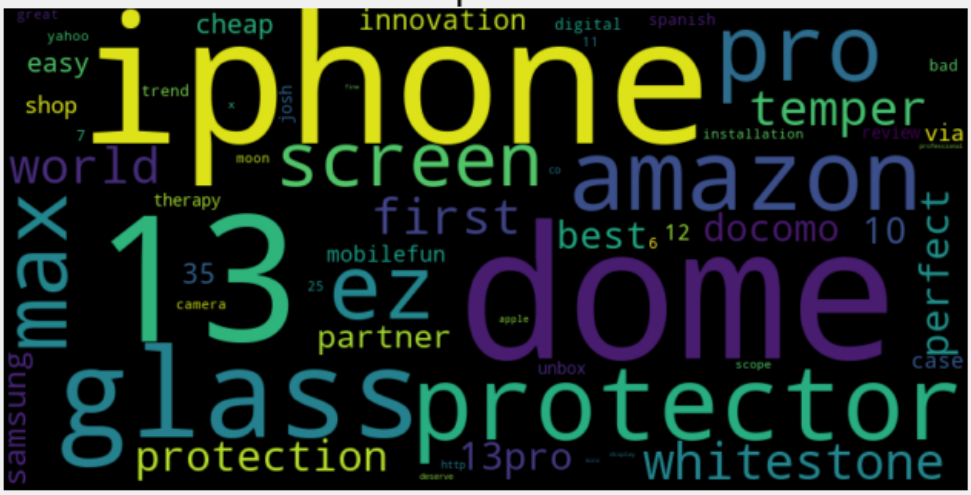


Figura 2.18: *Wordcloud* per ogni cluster

Il cluster 0 e il cluster 4 sono composti da poche unità (come si nota dalla rappresentazione grafica della distribuzione dei cluster) e presentano quindi pochi token "utili". Nel cluster 1, oltre ai token *iphone*, *13*, *pro* e *max*, spiccano le parole *case*, *protector* e *camera*: questo significa che i documenti inclusi nel cluster 1 fanno riferimento alle protezioni/coperture dell'iPhone, ma anche alla sua fotocamera. Anche nei cluster 2 e 5 si parla di *dome*, *glass*, *protection*, *whitestone* e *amazon*. In particolare, esiste il prodotto *Whitestone dome glass* che consiste in una pellicola protettiva per lo schermo dei telefoni. Il cluster 3 presenta dei termini più generali, tra cui *iphone*, *13*, *pro*, *apple* e *new*.

Regole di associazione all'interno dei cluster

Dopo aver trovato i token più frequenti all'interno dei 6 cluster, si vuole verificare l'esistenza di associazioni tra i token nei documenti.

Trovare delle **regole di associazione** significa trovare delle relazioni nascoste tra i dati. In questo contesto, questo risulta utile per dare un'interpretazione dei risultati, ma anche prevedere relazioni potenzialmente interessanti tra alcuni elementi e concetti. Una regola di associazione stabilisce un'implicazione tra due insiemi di elementi, pesata da una probabilità di un insieme condizionata all'altro insieme.

Considerando il gruppo di dati I e due sottoinsiemi di esso X e Y (chiamati anche *itemset*), la regola di associazione $X \Rightarrow Y$ è la coppia X, Y tali per cui

$$X \cap Y = \emptyset \quad X \neq \emptyset \quad Y \neq \emptyset$$

e

$$s(X \cup Y) \geq mins \quad c(X, Y) \geq minc$$

dove $s(X)$ è il *supporto* dell'*itemset* X, $c(X, Y)$ è la *confidenza* e *mins* e *minc* sono i rispettivi valori minimi.

Si definisce *transazione* su I un itemset $t = \{a_1, \ldots a_n\}$ dove $a_j \in I$.

$T = \{t_1, \ldots t_n\}$ è un insieme di n transazioni su I e *T(X)* è il sottoinsieme di transazioni che contengono X.

Il supporto di un *itemset* X è dato da

$$s(X) = \frac{|T(X)|}{|T|}$$

mentre la confidenza da

$$c(X, Y) = \frac{s(X \cup Y)}{s(X)}$$[16]

Per trovare le regole di associazione tra i token all'interno di ogni cluster, si usa l'algoritmo *Apriori*. L'algoritmo richiede il dataframe che contiene l'informazione sui token presenti in ogni documento. Si costruisce il dataframe `DataFrameBool`, che presenta nell'intersezione *(i,j)* il valore 0 se il token j non è presente nel documento i, o il valore 1 se invece è presente.

```
df1.column2=1
DataFrameBool = np.zeros((len(DataFrame),len(
   DataFrameTotalFreq)))
DataFrameBool = pd.DataFrame(DataFrameBool,columns
    = list(DataFrameTotalFreq.Token))
for i in range(len(df1)):
    DataFrameBool[(df1.column1[i])[1]][(df1.column
       1[i])[0]] = df1.column2[i]
DataFrameBool['Cluster']=kmeans.labels_
DataFrameBool

## Out:
```

[16]Melucci M. *Lezioni di Basi di Dati*. Padova: C.L.E.U.P., 2020

	0	000	1	10	100	1000	10000	100th	101	10th	...	zimbabwe	zip	zipper	zombie	zoom	□	□	□		Cluster
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10451	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
10452	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
10453	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	3
10454	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	3
10455	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3

10456 rows × 6114 columns

Figura 2.19: Dataframe *DataFrameBool*

Il codice che segue produce la tabella `RULES`, che presenta le regole di associazione trovate con la funzione `apriori` della libreria `mlxtend` e del pacchetto `frequent_patterns`.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_
    rules

df = apriori(DataFrameBool[DataFrameBool.Cluster==
    0].drop('Cluster',axis=1), min_support=0.5,use_
    colnames=True)
RULES = association_rules(df, metric="lift", min_
    threshold=1)
RULES['Cluster']=0
for i in range(1, knee):
    df = apriori(DataFrameBool[DataFrameBool.
        Cluster==i].drop('Cluster',axis=1), min_
        support=0.5,use_colnames=True)
    Rules = association_rules(df, metric="lift",
        min_threshold=1)
    Rules['Cluster']=i
    RULES= pd.concat([RULES, Rules], axis=0)

Counter(RULES.Cluster)
## Out: Counter({0: 6, 1: 10, 2: 213158, 3: 2, 4:
    2, 5: 229182})
```

Con un supporto minimo fissato del valore di 0.5, si trovano 6 regole di associazione tra i token all'interno del cluster 0, 10 all'interno del cluster 1, 213 158 nel cluster 2, 2 nel cluster 3 e 4, 229 182 nel cluster 5.

```
RULES[RULES.Cluster==0]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(13)	(pro)	0.812500	0.812500	0.812500	1.000000	1.230769	0.152344	inf	0
1	(pro)	(13)	0.812500	0.812500	0.812500	1.000000	1.230769	0.152344	inf	0
2	(13, iphone)	(pro)	0.804688	0.812500	0.804688	1.000000	1.230769	0.150879	inf	0
3	(pro, iphone)	(13)	0.804688	0.812500	0.804688	1.000000	1.230769	0.150879	inf	0
4	(13)	(pro, iphone)	0.812500	0.804688	0.804688	0.990385	1.230769	0.150879	20.3125	0
5	(pro)	(13, iphone)	0.812500	0.804688	0.804688	0.990385	1.230769	0.150879	20.3125	0

Figura 2.20: Regole di associazione nel cluster 0

```
RULES[RULES.Cluster==1]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(13)	(iphone)	0.976999	0.960570	0.948248	0.970572	1.010413	0.009772	1.339884	1
1	(iphone)	(13)	0.960570	0.976999	0.948248	0.987172	1.010413	0.009772	1.793063	1
2	(13)	(pro)	0.976999	0.603505	0.598302	0.612388	1.014719	0.008679	1.022917	1
3	(pro)	(13)	0.603505	0.976999	0.598302	0.991379	1.014719	0.008679	2.668127	1
4	(13, pro)	(iphone)	0.598302	0.960570	0.576396	0.963387	1.002933	0.001686	1.076944	1
5	(13, iphone)	(pro)	0.948248	0.603505	0.576396	0.607854	1.007207	0.004124	1.011092	1
6	(pro, iphone)	(13)	0.578039	0.976999	0.576396	0.997158	1.020633	0.011653	8.092552	1
7	(13)	(pro, iphone)	0.976999	0.578039	0.576396	0.589966	1.020633	0.011653	1.029088	1
8	(pro)	(13, iphone)	0.603505	0.948248	0.576396	0.955082	1.007207	0.004124	1.152146	1
9	(iphone)	(13, pro)	0.960570	0.598302	0.576396	0.600057	1.002933	0.001686	1.004387	1

Figura 2.21: Regole di associazione nel cluster 1

```
RULES[RULES.Cluster==2]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(amazon)	(13)	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	inf	2
1	(13)	(amazon)	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	inf	2
2	(dome)	(13)	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	inf	2
3	(13)	(dome)	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	inf	2
4	(13)	(ez)	1.000000	0.877506	0.877506	0.877506	1.000000	0.000000	1.000000	2
...	...	...	...	...	...	...	...	...	...	...
213153	(world)	(amazon, protector, max, temper, glass, iphone...	0.728285	0.601336	0.601336	0.825688	1.373089	0.163392	2.287071	2
213154	(dome)	(amazon, protector, max, temper, glass, iphone...	1.000000	0.601336	0.601336	0.601336	1.000000	0.000000	1.000000	2
213155	(13)	(amazon, protector, max, temper, glass, iphone...	1.000000	0.601336	0.601336	0.601336	1.000000	0.000000	1.000000	2
213156	(pro)	(amazon, protector, max, temper, glass, iphone...	0.819599	0.601336	0.601336	0.733696	1.220109	0.108482	1.497023	2
213157	(first)	(amazon, protector, max, temper, glass, iphone...	0.728285	0.601336	0.601336	0.825688	1.373089	0.163392	2.287071	2

213158 rows × 10 columns

Figura 2.22: Regole di associazione nel cluster 2

```
RULES[RULES.Cluster==3]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(13)	(iphone)	0.638455	0.738712	0.628498	0.984404	1.332595	0.156863	16.753938	3
1	(iphone)	(13)	0.738712	0.638455	0.628498	0.850802	1.332595	0.156863	2.423253	3

Figura 2.23: Regole di associazione nel cluster 3

```
RULES[RULES.Cluster==4]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(13)	(iphone)	1.0	1.0	1.0	1.0	1.0	0.0	inf	4
1	(iphone)	(13)	1.0	1.0	1.0	1.0	1.0	0.0	inf	4

Figura 2.24: Regole di associazione nel cluster 4

```
RULES[RULES.Cluster==5]

## Out:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	Cluster
0	(amazon)	(13)	0.869565	1.000000	0.869565	1.000000	1.000000	0.000000	inf	5
1	(13)	(amazon)	1.000000	0.869565	0.869565	0.869565	1.000000	0.000000	1.000000	5
2	(13)	(case)	1.000000	0.684783	0.684783	0.684783	1.000000	0.000000	1.000000	5
3	(case)	(13)	0.684783	1.000000	0.684783	1.000000	1.000000	0.000000	inf	5
4	(cheap)	(13)	0.521739	1.000000	0.521739	1.000000	1.000000	0.000000	inf	5
...	...	...	...	...	...	...	...	...	...	...
229177	(mobilefun)	(amazon, case, docomo, shop, perfect, iphone, ...	0.663043	0.521739	0.521739	0.786885	1.508197	0.175803	2.244147	5
229178	(dome)	(amazon, case, docomo, shop, perfect, iphone, ...	0.967391	0.521739	0.521739	0.539326	1.033708	0.017013	1.038176	5
229179	(whitestone)	(amazon, case, docomo, shop, perfect, iphone, ...	0.804348	0.521739	0.521739	0.648649	1.243243	0.102079	1.361204	5
229180	(13)	(amazon, case, docomo, shop, perfect, iphone, ...	1.000000	0.521739	0.521739	0.521739	1.000000	0.000000	1.000000	5
229181	(protection)	(amazon, case, docomo, shop, perfect, iphone, ...	0.815217	0.521739	0.521739	0.640000	1.226667	0.096408	1.328502	5

229182 rows × 10 columns

Figura 2.25: Regole di associazione nel cluster 5

Per ogni cluster, le regole di associazione che legano i token *iphone* a 13 o *pro*, e viceversa, risultano piuttosto scontate. Più interessanti, nel cluster 2, sono quelle dove compaiono i token *amazon* e *dome*, o nel cluster 5 *case* e *cheap*.

2.6 Sentiment Analysis

Si giunge ora alla parte conclusiva della procedura implementata, quella riguardante la *Sentiment Analysis*.

La *Sentimenti Analysis* consiste nell'identificare e categorizzare le opinioni presenti in una porzione di testo, in questo caso nei documenti (quindi nei tweet, nei post e nei commenti), di coloro che li hanno scritti.

Per fare questo, è necessario utilizzare la funzione `SentimentIntensityAnalyzer` presente nella libreria `nltk`.

```
from nltk.sentiment.vader import
    SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()
```

Dando come input alla funzione `polarity_scores` il contenuto testuale di ogni documento, viene restituito, sotto il nome di `compound`,

un numero intero che rappresenta la "forza" del sentimento.[17] All'interno dei documenti, i token vengono classificati come "positivi", "negativi" o "neutri", a seconda del sentimento che vanno ad esprimere. Se il numero restituito da `compound` è maggiore di 0, significa che nel documento a cui si riferisce la percentuale di token ritenuti positivi è maggiore rispetto a quella dei token ritenuti negativi; viceversa, è negativo se i token ritenuti negativi sono di più rispetto a quelli positivi. Il risultato poi può essere più o meno elevato a seconda della differenza tra numero di token ritenuti positivi e numero di token ritenuti negativi. Quando il risultato vale zero, il numero di token positivi e negativi si equivale.

```
DataFrame['Scores'] = DataFrame['ContentText'].
   apply(lambda Text: sia.polarity_scores(Text))
DataFrame['Compound']  = DataFrame['Scores'].apply
   (lambda s: s['compound'])
DataFrame['Label'] = DataFrame['Compound'].apply(
   lambda n: 'pos' if n >0 else 'neg' if n<0 else
   'neutral')

DataFrame['Pos'] = DataFrame['Scores'].apply(
   lambda s : s['pos'])
DataFrame['Neg'] = DataFrame['Scores'].apply(
   lambda s : s['neg'])
DataFrame['Neu'] = DataFrame['Scores'].apply(
   lambda s : s['neu'])
DataFrame["Cluster"] = kmeans.labels_

DataFrame['Scores']
## Out:
       0 {'neg': 0.0, 'neu': 0.444, 'pos': 0.556,
          ...
       1 {'neg': 0.0, 'neu': 0.545, 'pos': 0.455,
          ...
       2 {'neg': 0.0, 'neu': 0.444, 'pos': 0.556,
          ...
```

[17] NLTK Documentation, *nltk.sentiment.vader module*, https://www.nltk.org/api/nltk.sentiment.vader.html?highlight=polarity$_$scores

```
       3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, ...
       4 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, ...
                                  ...
   10451 {'neg': 0.0, 'neu': 0.894, 'pos': 0.106,
      ...
   10452 {'neg': 0.0, 'neu': 0.722, 'pos': 0.278,
      ...
   10453 {'neg': 0.0, 'neu': 0.705, 'pos': 0.295,
      ...
   10454 {'neg': 0.0, 'neu': 0.84, 'pos': 0.16,
      ...
   10455 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, ...
Name: Scores, Length: 10456, dtype: object

DataFrame['Pos']
## Out:
       0        0.556
       1        0.455
       2        0.556
       3        0.000
       4        0.000
                ...
   10451        0.106
   10452        0.278
   10453        0.295
   10454        0.160
   10455        0.000
Name: Pos, Length: 10456, dtype: float64

DataFrame['Neg']
## Out:
       0        0.0
       1        0.0
       2        0.0
       3        0.0
       4        0.0
                ...
   10451        0.0
   10452        0.0
```

```
    10453        0.0
    10454        0.0
    10455        0.0
Name: Neg, Length: 10456, dtype: float64

DataFrame['Neu']
## Out:
        0        0.444
        1        0.545
        2        0.444
        3        1.000
        4        1.000
                 ...
    10451        0.894
    10452        0.722
    10453        0.705
    10454        0.840
    10455        1.000
Name: Neu, Length: 10456, dtype: float64
```

Vengono quindi classificati i documenti a seconda dei risultati precedentemente descritti. Un documento che presenta il valore `compound` maggiore di 0 viene classificato come "positivo", se presenta un valore minore di 0 viene classificato come "negativo", se presenta il valore 0 come "neutro".

Vengono riportate le frequenze assolute dei documenti, classificati a seconda del sentimento per ogni cluster.

```
Labels=[]
for i in range(knee):
    Labels.append(DataFrame.Label[DataFrame.
       Cluster==i].value_counts())
Labels=pd.DataFrame(Labels)
Labels.index=range(knee)
Labels=Labels.fillna(0)

Labels
## Out:
```

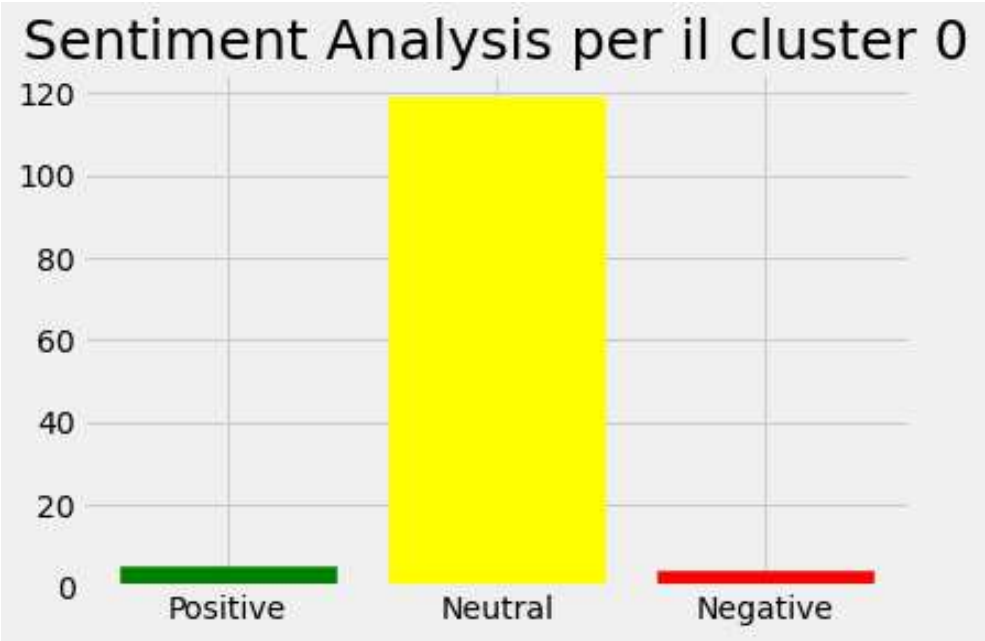
	neutral	pos	neg
0	119.0	5.0	4.0
1	1766.0	1416.0	470.0
2	0.0	449.0	0.0
3	1541.0	3235.0	1049.0
4	305.0	3.0	2.0
5	1.0	91.0	0.0

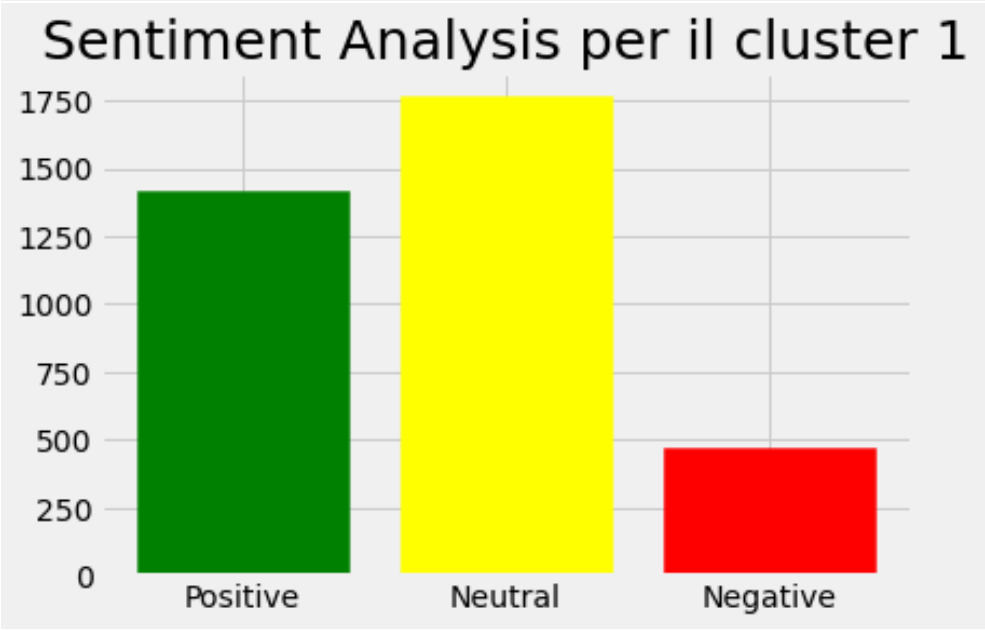
Figura 2.26: Frequenze dei documenti classificati a seconda del sentimento per ogni cluster

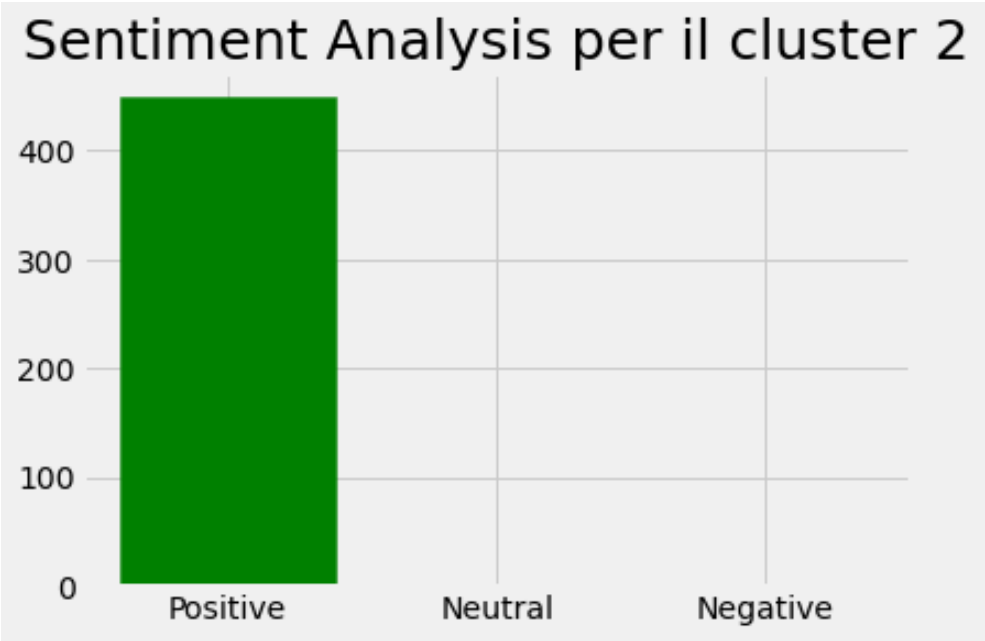
Viene facile rappresentare graficamente i risultati appena ottenuti tramite dei diagrammi a barre. L'asse delle ordinate indica la frequenza assoluta dei documenti classificati come, rispettivamente, "positivi", "negativi" e "neutri".

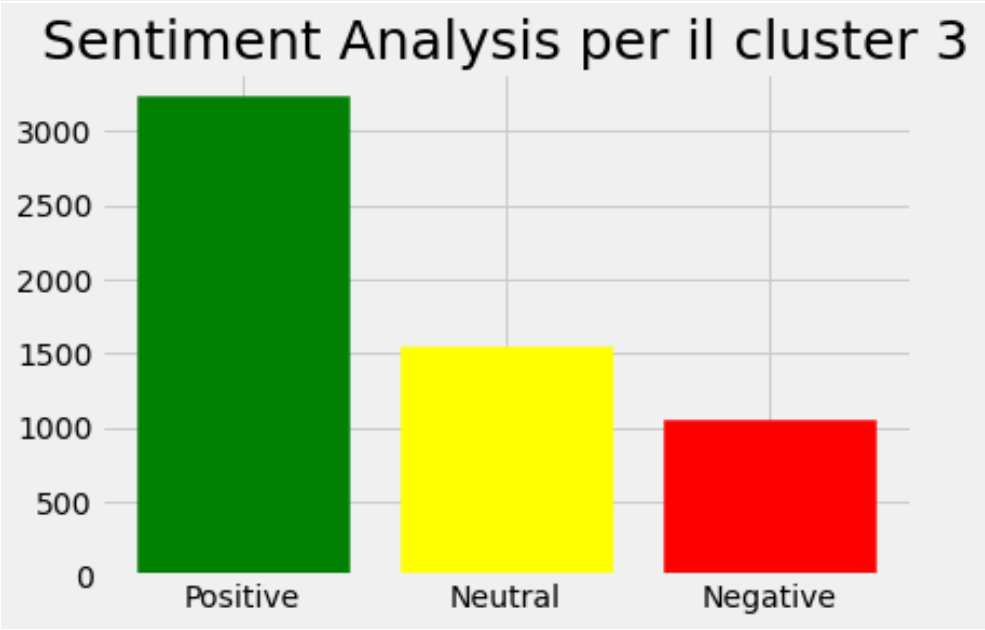
```
for i in range(knee):
    barlist=plt.bar(x=["Positive", "Neutral", "
       Negative"], height=[Labels['pos'][i],Labels
       ['neutral'][i],Labels['neg'][i]])
    barlist[0].set_color('green')
    barlist[1].set_color('yellow')
    barlist[2].set_color('red')
    plt.show()

## Out:
```


Sentiment Analysis per il cluster 0
120
100
80
60
40
20
0
Positive
Neutral
Negative


Sentiment Analysis per il cluster 1
1750
1500
1250
1000
750
500
250
0
Positive
Neutral
Negative


Sentiment Analysis per il cluster 2
400
300
200
100
0
Positive
Neutral
Negative


Sentiment Analysis per il cluster 3
3000
2500
2000
1500
1000
500
0
Positive
Neutral
Negative

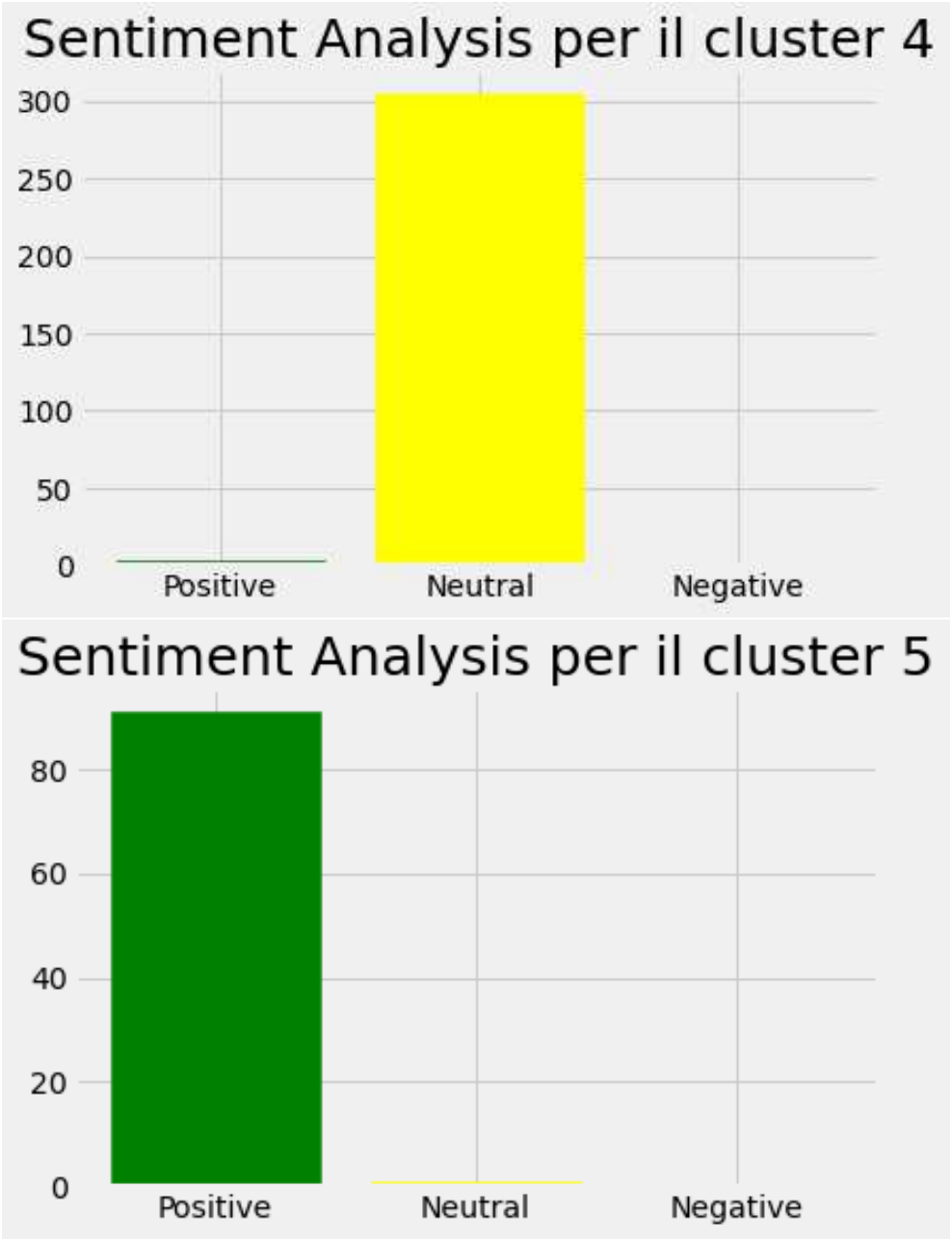


Figura 2.29: Diagramma a barre del sentimento per ogni cluster

Nei cluster 2, 3 e 5 prevale un sentimento positivo, che corrisponde infatti, rispettivamente, al 100%, al 55.54% e al 98.91% dei documenti totali presenti in questi cluster. Per i cluster 0, 1 e 4 vi è una prevalenza di giudizi neutri, con una percentuale di, rispettivamente, 92.97%, 48.36% e 98.39% sui documenti totali.

Il numero di documenti classificati negativi è sempre, comunque, molto basso rispetto agli altri: nei cluster 2 e 5, i commenti negativi

sono addirittura assenti.

2.7 Risultati

La procedura presentata permette dunque di raggiungere l'obiettivo prefissasto per il progetto, cioè quello di esaminare la valutazione generale degli utenti dei social network su un determinato prodotto, ma non solo; è stato possibile esplorare il dataset per valutare la presenza di possibili relazioni tra variabili, inoltre la divisione dei contenuti in cluster, la loro caratterizzazione e l'individuazione delle regole di associazione consentono di arricchire l'informazione e interpretare i risultati ottenuti dalla semplice classificazione dei giudizi.

L'analisi delle variabili numeriche presenti nel dataset ha evidenziato una correlazione positiva piuttosto elevata tra le variabili indicanti il numero di *followers* di un utente e di liste al quale esso fa parte, e una correlazione discreta tra il numero di "preferiti" di un tweet e il numero di retweet dello stesso. Entrambe le relazioni sono plausibili, in quanto entrambe sono sintomi di popolarità di un utente, nel primo caso, e di un tweet, nel secondo: all'aumentare del numero di seguaci di un utente, questo verrà inserito in più liste, mentre all'aumentare di "mi piace" di un tweet aumentano facilmente anche i retweet. Nonostante questo, l'analisi delle variabili numeriche non ha portato in rilievo risultati particolarmente significativi, a dimostrazione del fatto che, con questo tipo di dati, è più opportuno concentrarsi sull'analisi testuale.

Dopo aver attuato le procedure preliminati al *clustering* e aver assegnato un "peso" ai documenti con il metodo *tf-idf*, questi sono stati raggruppati in 6 cluster, sufficientemente differenziati tra di loro, nonostante un basso valore dell'indice *Silhouette*. Il primo cluster, o cluster 0, comprende documenti che contengono pochi token, così come anche il cluster 4. Di conseguenza, anche le regole di associazione trovate per entrambi i gruppi risultano essere

poco informative, dimostrando con una confidenza totale che, ogni qualvolta in un documento si trova il token *iPhone*, si trova anche il token *13* (e viceversa), associazione banale considerando il modo in cui è stata costruita la procedura. Di conseguenza, anche il sentimento "neutrale" degli utenti emerso con la *Sentiment Analysis* non suggerisce una vera e propria posizione degli utenti.

Nel cluster 1 tra i token maggiormente presenti, oltre i costanti *iphone*, *13*, *pro* e *max*, ci sono *case*, *apple* e *camera*, token che però non appaiono come elementi nelle regole di associazione. Anche in questo caso il sentimento degli utenti è principalmente "neutrale", tuttavia il numero di documenti classificati come "positivi" risulta comunque alto, con un 38.77% di opinioni positive sul totale. Queste possono essere relative ad un buon funzionamento della fotocamera oppure a un buon prodotto protettivo per il cellulare (in quanto token molto presenti all'interno del cluster) anche se, con le poche informazioni a disposizione, risulta difficile interpretare in modo ottimale il risultato.

Nel cluster 2 il token presente il maggior numero di volte è *dome*, seguito da *glass*, *protector*, *amazon* e *screen*. Anche le numerose regole di associazione sottolineano un forte legame tra questi token, che suggerisce che nel presente cluster si faccia riferimento ad una pellicola protettiva per lo schermo, venduta su *Amazon*. *Glass dome* significa, letteralmente, "cupola di vetro". È plausibile che all'interno di questo cluster siano presenti comunque molti tweet promozionali, nonostante lo sforzo fatto nelle prime fasi di raccolta dati per eliminarli il più possibile. Il sentimento degli utenti nel cluster 2 è totalmente positivo, un altro indizio che porterebbe a pensare che ci sia molta pubblicità.

Anche nel cluster 5 si fa riferimento ai token *dome*, *case*, *protection* e *amazon*, a cui si aggiungono *whitestone* e *perfect*. Si vedono associati i token *cheap* (che significa "economico") e *13* con una confidenza del 100%, ovvero ogni qualvolta è presente la parola *cheap*, è presente anche il numero *13*, connesso ovviamente con *iPhone 13*. Ulteriori regole di associazione accostano *mobilefun*,

un'azienda che offre diversi accessori per cellulari [18], con diversi token relativi alla vendita online di protettori schermo per telefoni, tra cui *amazon*, *case*, *shop* e *docomo*, altra azienda giapponese che si occupa di telefonia mobile. [19] Anche in questo cluster, i commenti sono quasi interamente positivi.

Il cluster 3 è caratterizzato dalla presenza di token più "generici" riguardo il *product* preso in riferimento, per esempio *iphone13*, *phone*, *new*, *apple*. Dalla *wordcloud* relativa si notano anche le parole *christmas*, *pixel*, *android*, rappresentate con una grandezza minore poiché meno frequenti. Le regole di associazione non suggeriscono relazioni significative tra i token con confidenza maggiore del 50%. Per i token presenti all'interno di questo cluster, sembra che questo corrisponda a quello in cui si parla più in generale del prodotto *iPhone 13*. La *Sentiment Analysis* riflette questo aspetto, infatti evidenzia un numero nettamente maggiore di documenti classificati come "positivi", seguito dai commenti "neutrali" e "negativi" che rimangono comunque di un numero alto.

Dalle analisi svolte, si può dunque affermare che l'*iPhone 13* è un prodotto generalmente apprezzato dagli utenti, anche se risulta difficile stabilire con esattezza i suoi punti di forza o di debolezza.

[18] MobileFun, `https://www.mobilefun.co.uk/`

[19] Wikipedia, *NTT docomo*, `https://it.wikipedia.org/wiki/NTT$\_$docomo`

Conclusione e considerazioni personali

I risultati ottenibili con la procedura costruita durante l'esperienza di stage appaiono qualitativamente molto differenti a seconda del *product* di riferimento scelto, ma anche rispetto al periodo in cui viene effettuata la ricerca. Come detto in precedenza, l'*iPhone 13* è un prodotto annunciato e messo sul mercato nel settembre del 2021. Nelle settimane successive al rilascio è apparso come un argomento di tendenza, quindi molto ricco e interessante da esplorare, ma progressivamente la sua popolarità è calata e, di conseguenza, anche i dati e la qualità delle informazioni disponibili (aspetto che rischia di creare "rumore" nell'esito dell'analisi). Per questo, con i risultati riportati nel presente elaborato è difficile individuare le qualità e i difetti riscontrati nell'*iPhone 13* che hanno portato gli utenti ad esprimere un giudizio positivo o negativo a riguardo, ma la procedura ha comunque dimostrato di essere valida e solida per qualsiasi prodotto d'interesse.

Il sistema implementato può ora rispondere all'esigenza di un'azienda, un ente o un singolo di esaminare, ma anche monitorare, l'andamento dell'opinione degli utenti riguardo un determinato prodotto, potendo anche scegliere se investire su ciò che risulta essere più o meno apprezzato.

Sono molto orgogliosa di poter dire che l'esperienza lavorativa fatta con l'azienda *Omicron Consulting* è stata davvero positiva e stimolante. Ho avuto modo di mettere in pratica le competenze coltivate durante gli anni di studio, ma anche di sperimentare con tecnologie e

funzionalità a me completamente nuove, come per esempio l'accesso ai social network da *RStudio* o il linguaggio di programmazione *Python*, che ho approcciato per la prima volta proprio durante questa esperienza. Vedere infine come l'impegno abbia permesso la realizzazione di un progetto concreto, mi ha regalato una grande soddisfazione.

Riconoscimenti

Durante lo stage ho avuto modo di lavorare individualmente, ma il lavoro di gruppo è stato fondamentale per il raggiungimento dell'obiettivo prefissato. Infatti, il risultato ottenuto a fine stage è stato il frutto della collaborazione con un altro stagista, Jacopo Dominici, con cui ho condiviso questa esperienza.

Durante l'esperienza lavorativa ho sempre avuto modo di confrontarmi con tutti i colleghi del gruppo di lavoro della sezione *Advanced Analytics*, in particolare con Sergio Talente, il responsabile della sezione che mi ha introdotto il progetto e tutti gli strumenti e le piattaforme dell'azienda, e Stefano Amodeo, il mio tutor aziendale, che è stato per me un punto di riferimento per tutta la durata dello stage.

Bibliografia e sitografia

[1] Melucci M. *Lezioni di Basi di Dati*. Padova: C.L.E.U.P., 2020.

[2] Johnson R.A.; Wichern D.W. *Applied Multivariate Statistical Analysis*. Pearson Education Limited, 2014, Sixth Edition

[3] Dispense del corso Analisi dei Dati Multidimensionali

[4] Documentazione di R

[5] Dean B., Backlinko, Backlinko, *How Many People Use Twitter in 2022? [New Twitter Stats]*, `https://backlinko.com/twitter-users`

[6] Stentella U., Lega Nerd, *Reddit ha svelato per la pri- ma volta quanti utenti ha*, `https://leganerd.com/2020/12/04/reddit-ha-svelato-per-la-prima-volta-quanti-utenti-ha/`

[7] Omicron Consulting, `https://www.omicronconsulting.it/approccio-e-mercati/` e `https://www.omicronconsulting.it/bi-advanced-analytics/`

[8] The RapidAPI, *Endpoint - What is an API Endpoint?*, `https://rapidapi.com/blog/api-glossary/endpoint/`

[9] Centro Assistenza Twitter, *Informazioni sulle API di Twitter*, `https://help.twitter.com/it/rules-and-policies/twitter-api`

[10] Twitter Developer Platform, *How to get access to the Twitter API*, `https://developer.twitter.com/en/docs/twitter-api/getting-started/getting-access-to-the-twitter-api`

[11] Twitter Developer Platform, *Counting characters*, `https://developer.twitter.com/en/docs/counting-characters`

[12] Documentazione Microsoft, *Scaricare SQL Server Manage- ment Studio (SSMS)*, `https://docs.microsoft.com/it-it/sql/`

ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

[13] NLTK Documentation, https://www.nltk.org/

[14] Analytics Vidha, *How to Get Started with NLP – 6 Unique Methods to Perform Tokenization*, https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/

[15] Manning C.D.; Raghavan P.; Schütze H. *Introduction to Information Retrieval, Stemming and lemmatization*, https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[16] OkPedia, *TF-IDF*, https://www.okpedia.it/tf-idf

[17] Bonaros B. Predictive Hacks, *K-Means Elbow Method Code For Python*, https://predictivehacks.com/k-means-elbow-method-code-for-python/

[18] NLTK Documentation, nltk.sentiment.vadermodule, https://www.nltk.org/api/nltk.sentiment.vader.html?highlight=polarity$_$scores

[19] MobileFun, https://www.mobilefun.co.uk/

[20] Wikipedia, NTT docomo, https://it.wikipedia.org/wiki/NTT$_$docomo

Elenco delle figure