Analysis of topics about cryptocurrencies on Reddit

Riccardo Ferrarese

19/3/2021

Contents

Introduction

Initial Questions

- Who are the most active users (in terms of number of posts and length of the messages)?
- How did these emotions change over time?

Dataset Issues

Setup App

Per la gestione del progetto sono state utilizzati i seguenti pacchetti:

- installare *git*, sistema per il controllo del versionamento
- installare *renv*, permette di creare un ambiente di esecuzione isolato per la gestione delle dipendenze. Permette anche di utilizzare il pacchetto *reticulate* per integrare degli script in Python e gestire l'ambiente locale per la loro esecuzione.

Per eseguire il progetto in R, dopo aver clonato il repository da github, è necessario impostare la working directory per installare i pacchetti necessari e utilizzare le funzioni del pacchetto *renv*:

```
setwd("~./<clone-dir>/Project")
install.packages("renv")  # packages dependecies write on renv.lock

renv::restore()
```

Le librerie utilizzate per eseguire l'analisi dei dati estratti dalla piattaforma Reddit sono le seguenti:

```
# general lib for manipulate data
library(dplyr)
library(tidyr)
library(tidyverse)
library(sets)

# lib for side by side plot
library(patchwork)

# lib for network analysis
library(ggraph)
library(tidygraph)
library(lubridate)

# lib for text mining
library(tidytext)
library(quanteda)
library(quanteda.textplots)
library(stringdist)

# lib for sentiment dataset
library(textdata)

# lib for lemmatize words
library(textstem)

# lib for stemming words
library(hunspell)
library(SnowballC)

library(tidytable)
library(microbenchmark)
library(quanteda.textstats)
```

Per installare l'ambiente necessario ad eseguire lo script in python e per eseguire l'analisi con la piattaforma **spacy** è consigliato installare un ambiente con conda.

We will use conda to manage virtual environments. Create the environment from file in git:

```
conda env create -f environment.devev.yml --name dsproject
```

Type the next command for check the env and activate it:

```
conda env list
conda activate dsproject
```

For trouble use pip:

```
conda install -n dsproject pip
conda activate dsprojectc
```

Reddit

Reddit, a differenza della maggior parte dei social-network maggiormente utilizzati, non presenta un meccanismo per scegliere una cerchia di utenti con cui interagire (per intenderci, i followers su Twitter o Instagram), tuttavia il modo in cui è organizzato permette all'utente di interagire in diversi forums, ognuno di uno specifico argomento, nel quale interagirà con gli altri utenti che 'seguo' lo stesso *subreddit*. Inoltre sulla pagina generale si possono trovare i post più popolari di tutta la piattaforma.

Si può definire Reddit come un *social-network aggregator*, cioè è una piattaforma di discussione in cui gli utenti possono discutere o condividere informazioni, suddivisa in forums di specifici argomenti chiamati **subreddits**.

Ogni utente può eseguire un numero illimitato di interazioni con la piattaforma, dove le interazioni possono essere la scrittura di un post in uno specifico subreddit, commentare post e interazioni di altri utenti o esprimere la propria preferenza relativa a un certo post o commento. Se un certo post (a meno che non sia di un subreddit privato) riceve molti *downvotes* immediatamente crollerà sulla 'classifica dei post' e scoparirà dalla vista degli altri utenti. Al contrario se acquisisce una certa importanza potrà esser visualizzato nella pagina generale di reddit e raggiungendo così un numero maggiore di utenti.

Un ulteriore particolarità di questa piattaforma è che gli utenti sono allo stesso modo creatore di contenuti, consumatori e curatori delle informazioni in esso. Utilizza infatti un sistema a punteggio, tramite *upvotes* e *downvotes* da parte degli utenti, per determinare i contenuti e le discussioni con maggior interesse e che verrano mostrati nei primi contenuti della pagina.

Pur permettendo agli utenti di mantenere l'anonimicità utilizzando un nome utente a piacere, Reddit tiene traccia di tutte le attività di ogni profilo inclusi i post e i commenti effettuati.

Collect Data with Python

Per la raccolta dati è stato utilizzato Python dal momento che è presente la libreria *Pushshift* per l'estrazione dei dati di Reddit senza vincoli stringenti sulle richieste effettuate. E' stato utilizzato un wrapper delle Python Reddit API, **pmaw**, che permette di eseguire lo scrapping dei dati utilizzando il multithreading, in modo tale da rendere più efficente il processo di raccolta dei dati.

Le API mettono a disposizione una serie di funzioni per la ricerca dei post d'interesse, potendo specificare l'intervallo temporale e i subreddit in cui eseguire la ricerca. Inoltre, volendo estrarre da Reddit le informazioni relative a diversi subreddit è stata utilizzata la libreria *multiprocessing* per lanciare un *pool* di processi incaricati di eseguire la stessa funzione su dati diversi.

```
LIST_of_SUBREDDIT = [ 'dogecoin',
    'pancakeswap',
    'eth',
    'ethereum',
    'Bitcoin',
    'elon']

    start_epoch=int(dt.datetime(2020, 1, 1).timestamp())
    end_epoch=int(dt.datetime(2021, 2, 1).timestamp())

    nargs = []
    for x in LIST_of_SUBREDDIT:
        nargs.append( [start_epoch, end_epoch, 'submissions', x] )
        nargs.append( [start_epoch, end_epoch, 'comments', x]  )

    pool = Pool(16)
    pool.map(run, nargs)
    pool.close()
    pool.join()
```

La funzione *run* lanciata da *pool.map* istanzia un oggetto di classe Miner contente la chiave di autenticazione per accedere a Pushshift e i metadati delle informazioni da raccogliere. In questo modo vengono lanciati diversi **Miner**, ognuno per uno specifico subreddit, i quali eseguono la ricerca e il salvataggio dei dati in parallelo.

```
def run(args) -> pd.DataFrame:
    """
    Function for starts miner's' process

    Args:
        [start, end]: [temporal intervall where we would scrap data]
        [item]: element to scrap
    Returns:
        [type]: [description]
    """

    print(args)
    start, end, item, subreddit = args

    miner = Miner(start, end, item, subreddit)
    miner.perform_search()
    return miner.read_data()

class Miner(object):
    """ Class for Reddit Data Mining"""

    def __init__(self, start_epoch, end_epoch, func, subreddit) -> None:
        super().__init__()
        self.api = PushshiftAPI(rate_limit=100)
        self.start_time = start_epoch
        self.end_time= end_epoch
        self.subreddit = subreddit
        self.data = None
```

```
        self.func = func

    def read_data(self):
        return self.data

    def perform_search(self):
        item = self.func
        print(f'Start search {item}...')
        if item == 'submissions':
            df = self.search_save_sub(self.subreddit)
            self.data = df
        if item == 'comments':
            df = self.search_save_com(self.subreddit)
            self.data = df

    @timeit
    def search_save_sub(self, subreddit):
        api = self.api
        res_ = api.search_submissions(after=self.start_time,
                                      before=self.end_time,
                                      subreddit=subreddit,
                                      filter=COLS_SUB,
                                      #limit=2
                                      )
        data = pd.DataFrame([x for x in res_])
        data.to_csv(f"./data/{self.subreddit}_sub.csv")
        print(f"write {self.subreddit}_sub.csv")

    @timeit
    def search_save_com(self, subreddit):
        api = self.api
        res_ = api = self.api.search_comments(after=self.start_time,
                                      before=self.end_time,
                                      subreddit=subreddit,
                                      filter=COLS_COM,
                                      #limit=2
                                    )
        data = pd.DataFrame([x for x in res_])
        data.to_csv(f"./data/{self.subreddit}_com.csv")
        print(f"write {self.subreddit}_com.csv")
```

E' stato utilizzato un *rate-limit* per imposta un numero di richieste all'API pari a 100 al minuto, leggermente superiore al limite di default (1 richesta per secondo) ma leggermente inferiore al limite imposto dalle richieste al minuto massime che si possono effettuare senza subire un ritardo.

La funzione *decorator timeit* permette inoltre di avere una misura indicativa del tempo impiegato per ogni richiesta. In totale l'esecuzione ha impiegato ..., ma essendo eseguita in parallelo il tempo reale di esecuzione è stato di un paio di ore.

Di seguito è mostrato un estratto di output da terminale che si ottiene dopo l'esecuzione del programma.

```
args first process: [1577833200, 1614553200, 'submissions']
args second process: [1577833200, 1614553200, 'comments']
Start search comments...
```

```
Start search submissions...
1430174 results available in Pushshift
259899 results available in Pushshift
Checkpoint:: Success Rate: 36.00% - Requests: 100 - Batches: 10 - Items Remaining: 256299
Checkpoint:: Success Rate: 29.00% - Requests: 100 - Batches: 10 - Items Remaining: 1427274
Checkpoint:: Success Rate: 35.50% - Requests: 200 - Batches: 20 - Items Remaining: 253225
Checkpoint:: Success Rate: 27.50% - Requests: 200 - Batches: 20 - Items Remaining: 1424674
Checkpoint:: Success Rate: 33.00% - Requests: 300 - Batches: 30 - Items Remaining: 250824
Checkpoint:: Success Rate: 29.33% - Requests: 300 - Batches: 30 - Items Remaining: 1421374
Checkpoint:: Success Rate: 31.00% - Requests: 400 - Batches: 40 - Items Remaining: 248435
Checkpoint:: Success Rate: 30.75% - Requests: 400 - Batches: 40 - Items Remaining: 1417879

[...]

write ./doge_sub.csv
time: 194798997.85995483 ~~ 3,25h
```

Overview of Datasets

Alcune osservazioni sui dati: - tutti i dati testuali fanno parte di commenti scritti da persone su un social network - si hanno un enorme quantità di parole scritte male o senza significato - le regex non sono sufficienti - hunspell

```
Object for Analysis
- corpus: --def of copurs
. dfm: --def of dfm
- tokens; .. def of tokens

Functions for Analysis
- corpus.* works if word.* function for create some dataFrame, clean and counts word/ngrams
- df.* is for the first step of each analysis

From RawData -> to DataFrame + Comment_Corpus

From Comment_Corpus -> to DFM + TIDY for each n-grams {1,2,3}
   toks with or without stop word
   apply stemming or lemmatization

From DataFrame -> Post_Corpus
   todo: topic analysis


--> return clean df for ngrams

from tidy :
FROM Count & Plot -> to :
               - Sentiment Class
               - Words Sentiment
               - Words Net

from dfm:
   - test stats quanteda
   - wordcloud quanteda
```

Text Analysis

Pipeline and objects for analysis

Quanteda Library

Quanteda è pacchetto per R per manipolare e analizzare dati testuali in formato non-tidy, sviluppata da Kenneth Benoi, Kohei Watanabe e altri contributors. Definisce una serie di funzioni per applicare processi di NLP (Natural Language Processing) a partire da testi di qualsiasi tipo o documenti, fino all'analisi finale.

In questo progetto è stata utilizzata per creare i *corpus* a partire dai dati estratti da reddit contenenti il testo dei commenti. Per comodità di utilizzo del pacchetto, sono state utilizzate le funzioni per la creazione dei *tokens* a partire dai commenti non manipolati, utilizzando le funzioni built-in per la rimozione di punteggiatura, simboli e le stop-words generali.

Successivamente alla manipolazione e pulizia dei commenti è stata utilizzata la funzione *dfm* per creare la *document-feature matrix*, in modo tale da rendere compatibile il tipo del dato con il pacchetto *tidytext* per la continuare l'analisi. Viene messo a disposizione anche un pacchetto per effettuare un'analisi statistica del document-feature matrix.

Una funzione motlo utile messa a disposizione dal pacchetto è quella per calcolare il valore di tfidf per la matrice dei documenti. Tuttavia, in questo caso di studio, risulta poco utile nel contesto di un singolo subreddit dal momento che si considera come documento ogni singolo commento.

Potrebbe però esser utile per comparare l'insieme totale di commenti di un subreddit con il contenuto presente negli altri.

```
dfm.frequency <- function(.dfm, n = 50) {

    features_dfm <- textstat_frequency(.dfm, n)

    head(features_dfm, 10)

    # Sort by reverse frequency order
    features_dfm$feature <- with(features_dfm, reorder(feature,
        -frequency))

    plot <- features_dfm %>%
        ggplot(aes(x = feature, y = frequency)) + geom_point(size = 0.1) +
        scale_y_log10() + theme_classic() + theme(axis.text.x = element_blank())

    features_dfm$feature <- with(features_dfm, reorder(feature,
        -rank))
    # theme(axis.text.x = element_text(angle = 90, hjust = 1))

    plot2 <- features_dfm %>%
        ggplot(aes(x = feature, y = rank)) + geom_point(size = 0.01) +
        scale_y_log10() + theme_classic() + theme(axis.text.x = element_blank())

    print(plot + plot2)

    return(features_dfm)
}
```

Function for Text Mining

In questa sezione si mostrano le funzione definite per manipolare, pulire e visualizzare i dati estratti da reddit.

Si è scelto di definire delle funzioni il più generali possibili, in modo tale da rendere più efficiente l'analisi di diversi dataset, che condividono le stesse variabili esplicative ma riferiti a subereddit diversi. Le stesse funzioni possono inoltre esser utilizzate per uno sviluppo futuro di una *shiny app*.

La prima funzione definita , *df.create*, verrà utilizzata per creare il dataset di partenza a partire dai dati grezzi salvati in formato csv dallo script in Python. L'algoritmo esegue i seguenti passaggi: - converte in un formato comodo per R la colonna rappresentate le date di creazione dei commenti. - aggiunge un id per ogni riga. - rimuove i commenti cancellati o quelli che sono stati sottoposti ad una moderazione. - aggregare i testi dei commenti come post. - crea il corpus utilizzando la funzione messa a disposizione dal pacchetto *quanteda*. - salva i dataset in un oggetto R (.rds) per un utilizzo futuro

```
## data must have cols : (created_utc, author, body ) data is
## pushshift's returned csv
df.create <- function(data, dir, subR, filter_post = -1) {

    # compute type date add unique id for each comment filter
    # removed comment
    data_clean <- data %>%
        mutate(date = as.Date(as_datetime(created_utc))) %>%
        filter(author != "[deleted]" & author != "AutoModerator") %>%
        mutate(id = row_number()) %>%
        distinct(author, date, body, parent_id, link_id, .keep_all = TRUE) %>%
        select(id, date, author, body, parent_id, link_id)

    ## remove source data for save space
    remove(data)

    # DF for Corpus by POST
    df_post <- data_clean %>%
        select(link_id, body) %>%
        # aggregate comment whith same postID
    aggregate(by = list(data_clean$link_id), paste) %>%
        # compute num of element in body list for each postID
    mutate(n_com = unlist(map(link_id, length))) %>%
        rename(postID = "Group.1") %>%
        select(postID, body, n_com) %>%
        # group by post ID and merge senteces in body
    group_by(postID) %>%
        mutate(body = paste(do.call(c, body), collapse = "\n ")) %>%
        ungroup() %>%
        # filtra post con meno di 5 commenti
    filter(n_com > filter_post)

    post_corpus <- corpus(df_post$body, docnames = df_post$postID,
        docvars = data.frame(n = df_post$n_com))

    # quanteda corpus function
    corpus <- corpus(as.character(data_clean$body), docnames = data_clean$id,
        docvars = data.frame(author = data_clean$author, data = data_clean$date,
            link_id = data_clean$link_id))
```

```
    # print(summary(data_clean[ c('id', 'date', 'author')] ))

    rtn <- list(df_comm = data_clean, df_post = df_post, corpus_comm = corpus,
        corpus_post = post_corpus)
    saveRDS(rtn, paste0(dir, subR, ".rds", sep = ""))

    return(rtn)
}
```

Tokenize Corpus La preparazione iniziale dei dati ci permette di avere ai dati iniziali dell'analisi senza dover rielaborare il csv iniziale da zero. Le seguenti funzioni hanno l'obiettivo di calcolare i *tokens* per ciascun commento, costruendo quindi un *document-features matrix* permettendoci di: - analizzare il dizionario, effettuando delle analisi descrittive e osservare i cambiamenti dei dati dopo ciascuna manipolazione. - riflettere sulla bontà dei dati e scegliere delle tecniche per poter migliorare la correttezza delle parole.

La funzione che calcola i token permette di scegliere, tramite argomento, se calcolare i token rispetto singola parola oppure se calcolarli come bigrammi o trigrammi, in modo tale da poter eseguire un'analisi tenendo conto delle relazioni tra le diverse parole adiacenti in uno stesso commento.

Successivamente le parole vengono filtrate in modo tale da rimuove caratteri speciali che si trovano agli estremi di esse, infatti reddit permette di inserire dei caratteri per enfatizzare le parole, similmente all' *underscore* e altri nel formato Markdown. Inoltre vengono rimosse le parole formate da uno e due caratteri, in modo tale da ridurre il numero di onomatopee che vengono utilizzate nel linguaggio dei messaggi. Questa scrematura non è risultata sufficiente per costruire un dataset con solo parole con un significato compiuto, ma dal momento che 'parole' prive di significato avranno poche ripetizioni all'interno di tutto il corpus di testo, potranno esser rimosse successivamente dai grafici con un filtro sul conteggio. Inoltre per l'analisi del sentimento un gran numero di parole non troveranno un accoppiamento nei dataset messi a disposizioni, tra queste anche quelle prive di significato.

Dopo una prima pulitura delle parole è possibile, tramite argomento della funzione, specificare se si vuole o meno eseguire lo *stemming* oppure la *lemmatization*.

- sentiment with lammatization
- count with stemming

Function for prepare data Le seguenti funzioni hanno lo scopo di calcolare i *token* per ciascun commento e post, eseguire una pulizia di essi e calcolare le ripetizioni di ciascuna parola.

```
## function for normalize value between (0-1]
# x = data.frame(n = c(1,2,3,4,5,6,7))
# x %>% mutate(minmax = range01(n))
range01 <- function(x){
               min <- min({{x}})
               max <- max({{x}})
               (x-min+1)/(max-min+1)
}

## func for clean a little bit the words
# data for dyplr pipe --
#          curly-curly impl.  := to assign,
#                             {{}} to ref a cols in data,
#                             !!! access to the value
# data:: tidy text object
```

```r
# word:: ref to cols in data
word.apply_regexs <- function(data, word){

   ## use curly-curly for word cols
   data %>%
      mutate( {{word}} := str_extract({{word}},  "[a-z]+")) %>%
      # rimuove spazi ecc
      mutate( {{word}} := gsub("[[:punct:]]", "", {{word}})) %>%
      # word with all same char
      mutate( {{word}} := gsub("^([:alpha:])\1+", "", {{word}})) %>%
      mutate( {{word}} := gsub("^(a){5}[a-z]*", "", {{word}})) %>%
      # word with 1 or 2 char
      mutate( {{word}} := gsub("^[a-z]{1,2}\\b", "", {{word}})) %>%

      # word with numeber
      # mutate( word = gsub("^(\\d)+", "", word)) %>%

      # laughing word
      mutate( {{word}} := gsub("\\b(?:a*(?:ha)+h?|h*ha+h[ha]*|(?:l+o+)+l+|o?l+o+l+[ol]*)\\b", "", {{wor
      filter( {{word}} != "" )

}

## function for lemmatization, stemming or POS for words, bi-grams and tri-grams
# data:: tidy text object
# mode:: {"lemma", "stem", "other"} default is "none"
# words{1,2,3}:: ref to cols in data
word.manipulation <- function(data, word1, word2, word3, mode = 0, ngrams = 1){


   if(mode == 1){
      if(ngrams == 2){
         data %>%
            mutate({{word1}} := textstem::lemmatize_words({{word1}})) %>%
            mutate({{word2}} := textstem::lemmatize_words({{word2}}))
      } else if(ngrams == 3){
         data %>%
            mutate({{word1}} := textstem::lemmatize_words({{word1}})) %>%
            mutate({{word2}} := textstem::lemmatize_words({{word2}})) %>%
            mutate({{word3}} := textstem::lemmatize_words({{word3}}))

      }else {
         data %>%
            mutate( {{word1}} := textstem::lemmatize_words({{word1}}))

      }
   } else if(mode == 2){
      if(ngrams == 2){
         data %>%
            mutate({{word1}} := SnowballC::wordStem({{word1}}, language = "english")) %>%
            mutate({{word2}} := SnowballC::wordStem({{word2}}, language = "english"))
      } else if(ngrams == 3){
         data %>%
```

```
            mutate({{word1}} := SnowballC::wordStem({{word1}}, language = "english")) %>%
            mutate({{word2}} := SnowballC::wordStem({{word2}}, language = "english")) %>%
            mutate({{word3}} := SnowballC::wordStem({{word3}}, language = "english"))
      } else {
         data %>%
            mutate({{word1}} := SnowballC::wordStem({{word1}}, language = "english"))

      }

   } else {
      data
   }

}
```

La funzione che calcola i token permette di scegliere, tramite argomento, se calcolare i token rispetto singola parola oppure se calcolarli come bigrammi o trigrammi, in modo tale da poter eseguire un'analisi tenendo conto delle relazioni tra le diverse parole adiacenti in uno stesso commento.

Successivamente le parole vengono filtrate in modo tale da rimuove caratteri speciali che si trovano agli estremi di esse, infatti reddit permette di inserire dei caratteri per enfatizzare le parole, similmente all' *underscore* e altri nel formato Markdown. Inoltre vengono rimosse le parole formate da uno e due caratteri, in modo tale da ridurre il numero di onomatopee che vengono utilizzate nel linguaggio dei messaggi. Questa scrematura non è risultata sufficiente per costruire un dataset con solo parole con un significato compiuto, ma dal momento che 'parole' prive di significato avranno poche ripetizioni all'interno di tutto il corpus di testo, potranno esser rimosse successivamente dai grafici con un filtro sul conteggio. Inoltre per l'analisi del sentimento un gran numero di parole non troveranno un accoppiamento nei dataset messi a disposizioni, tra queste anche quelle prive di significato.

Dopo una prima pulitura delle parole è possibile, tramite argomento della funzione, specificare se si vuole o meno eseguire lo *stemming* oppure la *lemmatization*.

- sentiment with lammatization
- count with stemming

```
## tokenize corpus and make document-feature matrix (DFM +
## TIDY) data :: text corpus stop :: bool for remove stop word
## ngrams :: {1,2,3} for compute corpus on ngrams
corpus.tokenize_dfmTidy <- function(data, stop = TRUE, correct = FALSE,
    ngrams = 1) {

    # if stopword ..  build toks
    if (stop) {
        toks <- quanteda::tokens_select(quanteda::tokens(data,
            remove_punct = TRUE, remove_symbols = TRUE, remove_numbers = TRUE,
            remove_url = TRUE), pattern = stopwords("en"), selection = "remove")
    } else {
        toks <- quanteda::tokens(data, remove_punct = TRUE, remove_symbols = TRUE,
            remove_numbers = TRUE, remove_url = TRUE)
    }

    if (correct) {
        # spell checking on toks
    }
```

```
    # if ngrams -- compute toks for bigrams or trigrams
    if (ngrams == 2) {
        dfm <- quanteda::dfm(quanteda::tokens_ngrams(toks, n = 2,
            concatenator = " "))
    } else if (ngrams == 3) {
        dfm <- quanteda::dfm(quanteda::tokens_ngrams(toks, n = 3,
            concatenator = " "))
    } else {
        dfm <- quanteda::dfm(toks)
    }

    # return dfm and tidy
    return(list(dfm = dfm, tidy = tidy(dfm)))
}

## function for clean data and separate ngrams tidy ::
## dataframe compute by create corpus ( tidy dfm ) ngrams ::
## {1,2,3} for compute corpus on ngrams
corpus.clean_tidy <- function(tidy, ngrams = 1, mode = "none") {

    if (mode == "lemma") {
        .mode <- 1
    } else if (mode == "stem") {
        .mode <- 2
    } else {
        .mode <- 0
    }
    #### check ngrams and build clean dataframe
    if (ngrams == 2) {
        clean_df <- tidy %>%
            ## mutate term in word
        unnest_tokens(words, term, token = "ngrams", n = 2) %>%
            ## divide bigrams in words
        tidyr::separate(words, c("word1", "word2"), sep = " ",
            remove = FALSE) %>%
            filter(word1 != word2 & word2 != word1) %>%
            # apply some regex
        word.apply_regexs(word1) %>%
            word.apply_regexs(word2) %>%
            # apply stemming or lemmatiz ..
        word.manipulation(word1, word2, mode = .mode, ngrams = ngrams)

    } else if (ngrams == 3) {
        clean_df <- tidy %>%
            unnest_tokens(words, term, token = "ngrams", n = 3) %>%
            tidyr::separate(words, c("word1", "word2", "word3"),
                sep = " ", remove = FALSE) %>%
            filter(word1 != word2 & word2 != word3 & word1 !=
                word3) %>%
            word.apply_regexs(word1) %>%
            word.apply_regexs(word2) %>%
            word.apply_regexs(word3) %>%
            word.manipulation(word1, word2, word3, mode = .mode,
```

```
                ngrams = ngrams)

    } else {
        clean_df <- tidy %>%
            unnest_tokens(words, term) %>%
            word.apply_regexs(words) %>%
            word.manipulation(words, mode = .mode, ngrams = ngrams)
    }

    return(clean_df)
}

## function for word (or ngrams) counts and plot data :: clean
## df TIDY DFM

# ngrams :: {1,2,3} for compute corpus on ngrams
# threshold_count :: value to filter items by count for
# pritty plot threshold_freq :: value to filter items by
# frequency for pritty plot bool_plot_count :: boolean to
# indicate whether to plot word counts or not bool_plot_freq
# :: boolean to indicate whether to plot word frequencies or
# not
corpus.countPlot_tidy <- function(data, threshold_count = 1000,
    threshold_freq = 0.001, ngrams = 1, bool_plot_count = TRUE,
    bool_plot_frequency = TRUE) {

    if (ngrams == 2) {
        # merge single word
        words_unite <- data %>%
            unite(words, c("word1", "word2"), sep = " ")
        # sum count col for each word
        word_counts <- aggregate(cbind(count) ~ words, data = words_unite,
            FUN = sum)

    } else if (ngrams == 3) {
        words_unite <- data %>%
            unite(words, c("word1", "word2", "word3"), sep = " ")
        word_counts <- aggregate(cbind(count) ~ words, data = words_unite,
            FUN = sum)

    } else {
        word_counts <- aggregate(cbind(count) ~ words, data = data,
            FUN = sum)
    }

    total_of_word <- sum(word_counts$count)
    word_counts <- word_counts %>%
        mutate(count = as.integer(count)) %>%
        mutate(total_of_word = total_of_word) %>%
        mutate(frequency = count/total_of_word)

    # plot for count col
    plot_freq <- word_counts %>%
```

```
        filter(frequency > threshold_freq) %>%
        ggplot(aes(words, frequency)) + geom_point(alpha = 0.3,
        size = 1.5, width = 0.25, height = 0.1) + geom_text(aes(label = words),
        check_overlap = TRUE, vjust = 1) + theme_classic() +
        theme(axis.text.x = element_blank())

    # plot for frequency col
    plot_count <- word_counts %>%
        filter(count > threshold_count) %>%
        ggplot(aes(words, count)) + geom_point(alpha = 0.3, size = 1.5,
        width = 0.25, height = 0.1) + geom_text(aes(label = words),
        check_overlap = TRUE, vjust = 1) + scale_y_log10() +
        theme_classic() + theme(axis.text.x = element_blank())

    # print selected plot
    if (bool_plot_count & bool_plot_frequency) {
        print(plot_freq + plot_count + plot_layout(ncol = 1,
            heights = c(4, 4)))
    } else if (bool_plot_count) {
        print(plot_count)
    } else if (bool_plot_frequency) {
        print(plot_freq)
    }

    # split words for sentiment analysis
    if (ngrams == 3) {
        word_counts <- word_counts %>%
            tidyr::separate(words, c("word1", "word2", "word3"),
                sep = " ")
    } else if (ngrams == 2) {
        word_counts <- word_counts %>%
            tidyr::separate(words, c("word1", "word2"), sep = " ")
    }

    return(word_counts)

}
```

Di seguito è stata definita una funzione per visualizzare con un differente grafico la frequenza delle parole nei testi presenti in ciascun subreddit. Viene utilizza la funzione *textsta_frequency* messa a disposizione dalla libreria quanteda e a differenza della funzione sopradefinita, utilizza il document-feature matrix calcolato precedentemente senza passare per il formato tidy.

Verrà utilizzata anche la funzione *textplot_wordcloud* per rendere più piacevole e veloce il riconoscimento delle parole più frequenti.

Sentiment

Le funzioni definite per l'analisi del sentimento del testo permettono di classificare ciascuna parola (o n-gramma) secondo i valori built-in dei dataset messi a disposizione nella libreria *textdata*.

Per un analisi generale dei sentimenti di ciascun subreddit si è scelto di utilizzare i sentimenti elencati nel dataset *"nrc"*. Per classificare ciascuna parola sono stati utilizzati entrambe le categorizzazioni messe a disposizione da *"afinn"*, che permette di avere un punteggio numerico per identificare la positività o meno del sentimento, e da *"bing"*, che più semplicemente da solo una categorizzazione tra parole positive e negative.

L'ultima funzione permette di visualizzare la rete che rappresenta le relazioni che intercorrono tra le singole parole contenute nei bigrammi e trigrammi.

```
## function for print number of word for each sentiment in nrc df
# data :: word_frequency
# n_filter_sentiment :: value to filter number of sentiment for pritty plot
# ngrams :: {1,2,3} for compute corpus on ngrams
words.classSentiment <- function(data, n_filter_sentiment, ngrams=1){
   # inner join with nrc sentiment
   if(ngrams == 3 ){
      sentiment_class <- data %>%
         # join sentiment with each word
         inner_join(get_sentiments("nrc"), by=c('word1' = 'word')) %>%
         inner_join(get_sentiments("nrc"),by=c('word2' = 'word')) %>%
         inner_join(get_sentiments("nrc"),by=c('word3' = 'word')) %>%
         # merge sentiment
         unite(sentiment, c("sentiment.x", "sentiment.y", "sentiment"), sep="-") %>%
         # group and count
         group_by(sentiment) %>%
         summarise(word_4_sentiment = n()) %>%
         arrange(-word_4_sentiment, sentiment)

      # plot number of word for each sentiment class
      plot <- sentiment_class %>%
         filter(word_4_sentiment > n_filter_sentiment ) %>%
            ggplot(aes(word_4_sentiment, sentiment, fill=sentiment)) +
            geom_col( show.legend = FALSE) +
            xlab("") +
            theme_minimal()


   } else if(ngrams == 2 ){
      sentiment_class <- data %>%
         inner_join(get_sentiments("nrc"), by=c('word1' = 'word')) %>%
         inner_join(get_sentiments("nrc"),by=c('word2' = 'word')) %>%
         unite(sentiment, c("sentiment.x", "sentiment.y"), sep="-") %>%
         group_by(sentiment) %>%
         summarise(word_4_sentiment = n()) %>%
         arrange(-word_4_sentiment, sentiment)


      plot <- sentiment_class %>%
               filter(word_4_sentiment > n_filter_sentiment ) %>%
                  ggplot(aes(word_4_sentiment, sentiment, fill=sentiment)) +
                  geom_col( show.legend = FALSE) +
                  xlab("") +
                  theme_minimal()


   } else {
      sentiment_class <- data %>%
         inner_join(get_sentiments("nrc"), by=c('words'= 'word')) %>%
         group_by(sentiment) %>%
         summarise(word_4_sentiment = n()) %>%
```

```
        arrange(-word_4_sentiment, sentiment)

    plot <- sentiment_class %>%
             #filter(word_4_sentiment > 2500 ) %>%
                ggplot(aes(word_4_sentiment, sentiment, fill=sentiment)) +
                geom_col( show.legend = FALSE) +
                xlab("") +
                theme_minimal()
  }


  print(plot)
  return(sentiment_class)
}




## function for print words splited by sentiment
# data :: word_frequency
# n_filter :: value to filter words by count for pritty plot
# ngrams :: {1,2,3} for compute corpus on ngrams
words.computeSentiment <- function(data, n_filter=20, ngrams=1, plot = TRUE){

  if(ngrams == 2){
    ## with affin --> mean between value.x & value.y
    ## with bing
      ## positive - positive --> positive
      ## positive - negative --> neutral
      ## negative - positive --> neutral
      ## negative - negative --> negative

    sentiment_df <- data %>%
          inner_join(get_sentiments("afinn"), by=c('word1'= 'word')) %>%
          inner_join(get_sentiments("bing"), by=c('word1'= 'word')) %>%
          inner_join(get_sentiments("afinn"), by=c('word2'= 'word')) %>%
          inner_join(get_sentiments("bing"), by=c('word2'= 'word')) %>%
          mutate( affin = (value.x + value.y) / 2) %>%
          mutate( bing = case_when(sentiment.x == 'positive' &
                                     sentiment.y == 'positive' ~ 'positive',
                                   sentiment.x == 'negative' &
                                     sentiment.y == 'negative' ~ 'negative',
                                   TRUE ~ 'neutral')) %>%
          unite(words, c("word1", "word2"), sep = " ")

    if(plot){
    p <- sentiment_df %>%
      dplyr::filter( count > n_filter ) %>%
      # compute normalize value of sentiment
      mutate( affin_nrm = range01(affin)) %>%
      ggplot( aes(words, count,  color = affin_nrm)) +
        geom_jitter(alpha = 0.2, width=0.2, height = 0.1) +
```

```
        geom_text(aes(label = words), check_overlap = TRUE, vjust = 1.5) +
        scale_y_log10() +
        # split positive - negative
        facet_wrap(bing~.) +
        theme_minimal() +
        theme(axis.text.x=element_blank(),
              legend.position = "bottom") +
        labs( color = "Sentiment degree")
    }


} else if(ngrams == 3){

    sentiment_df <- data %>%
        inner_join(get_sentiments("afinn"), by=c('word1'= 'word')) %>%
        inner_join(get_sentiments("bing"), by=c('word1'= 'word')) %>%
        inner_join(get_sentiments("afinn"), by=c('word2'= 'word')) %>%
        inner_join(get_sentiments("bing"), by=c('word2'= 'word')) %>%
        inner_join(get_sentiments("afinn"), by=c('word3'= 'word')) %>%
        inner_join(get_sentiments("bing"), by=c('word3'= 'word')) %>%
        mutate( affin = (value.x + value.y + value) / 3) %>%
        mutate( bing = case_when(sentiment.x == 'positive' &
                                    sentiment.y == 'positive' &
                                    sentiment == 'positive' ~ 'positive',
                                 sentiment.x == 'negative' &
                                    sentiment.y == 'negative' &
                                    sentiment == 'negative' ~ 'negative',

                                 sentiment.x == 'positive' &
                                    sentiment.y == 'positive' &
                                    sentiment == 'negative' ~ 'neutral-positive',
                                 sentiment.x == 'positive' &
                                    sentiment.y == 'negative' &
                                    sentiment == 'positive' ~ 'neutral-positive',
                                 sentiment.x == 'negative' &
                                    sentiment.y == 'positive' &
                                    sentiment == 'positive' ~ 'neutral-positive',

                                 TRUE ~ 'neutral-negative')) %>%
        unite(words, c("word1", "word2", "word3"), sep = " ")

    if(plot){
    p <- sentiment_df %>%
       dplyr::filter( count > n_filter ) %>%
       mutate( affin_nrm = range01(affin)) %>%

       ggplot( aes(words, count,  color = affin_nrm)) +
          geom_jitter(alpha = 0.2, width=0.2, height = 0.1) +
          geom_text(aes(label = words), check_overlap = TRUE, vjust = 1.5) +
          scale_y_log10() +
          facet_wrap(bing~.) +
          theme_minimal() +
          theme(axis.text.x=element_blank(),
```

```
                legend.position = "bottom") +
          labs( color = "Sentiment degree")
      }
   } else {
   # plot positive-negative -- color: how much positive/negative is a word
      sentiment_df <- data %>%
            inner_join(get_sentiments("afinn"), by=c('words'= 'word')) %>%
            inner_join(get_sentiments("bing"), by=c("words" = "word"))

      if(plot){
      p <- sentiment_df %>%
         dplyr::filter( count > n_filter ) %>%
         mutate( value_std = range01(value)) %>%

            ggplot( aes(words, count,  color = value_std)) +
               geom_jitter(alpha = 0.2, width=0.2, height = 0.1) +
               geom_text(aes(label = words), check_overlap = TRUE, vjust = 1.5) +
               facet_wrap(sentiment~.) +
               scale_y_log10() +
               theme_minimal() +
               theme(axis.text.x=element_blank(),
                     legend.position = "bottom") +
               labs( color = "Sentiment degree")
      }
   }

   if(plot){ print(p) }
   return(sentiment_df)
}


words.network <- function(data, n_filter=1000){

   word_graph <- data %>%
      filter(count > n_filter) %>%
      as_tbl_graph()

   a <- grid::arrow(type = "open", length = unit(.1, "inches"))

   graph <- ggraph(word_graph, layout = "fr") +
        geom_edge_link(aes(edge_alpha = count), show.legend = FALSE,
                       arrow = a, end_cap = circle(.07, 'inches')) +
        geom_node_point(color = "lightblue", size = 1) +
        geom_node_text(aes(label = name), vjust = 1.5, hjust = 1) +
        theme_void()

   print(graph)
}
```

Topic modelling aggregando tutte i commenti di un certo post per ogni row con parent_id concateno tutti i body dei commenti con quel parent_id

https://gensimr.news-r.org/reference/model_lda.html

Dopo aver definito tutte le funzioni possiamo vedere come esse siano organizzate, così da poter capire il

codice delle analisi di ciascun dataset. ## DogeCoin's Subreddit

Overview of Dataset

Come prima cosa scarichiamo i dati dalla cartella del codice Python riguardanti il subreddit da analizzare.

```
data <- readRDS("../Data/dogecoin.rds")

print("REDDIT's COMMENT DF: ")
summary(data$df_comm[c("id", "date", "author")])
print("REDDIT's POST DF: ")
summary(data$df_post)
```

```
# raw corpus
com_doge <- corpus.tokenize_dfmTidy(data$corpus_comm)
print("REDDIT's COMMENT CORPUS: ")
print(com_doge$dfm)
post_doge <- corpus.tokenize_dfmTidy(data$corpus_post, stop = FALSE)
print("REDDIT's POST CORPUS: ")
print(post_doge$dfm)
```

```
print("stop words removed:")
77739 - 77567
```

Written texts show the remarkable feature that the rank-ordered distribution of word frequencies follows an approximate power law

Display Formula (Thurner et al. 2015) where r is the rank that is assigned to every word in the text. For most texts, regardless of language, time of creation, genre of literature, its purpose, etc. one finds that 1, which is referred to as Zipf's law [1].

```
print("CELLS of mem FOR DFM")
print(prod(dim(com_doge$dfm)))
print("SPARSITY FOR dfm")
print(sparsity(com_doge$dfm))
print("---")
print("CELLS of mem FOR post DFM")
print(prod(dim(post_doge$dfm)))
print("SPARSITY FOR post dfm")
print(sparsity(post_doge$dfm))
```

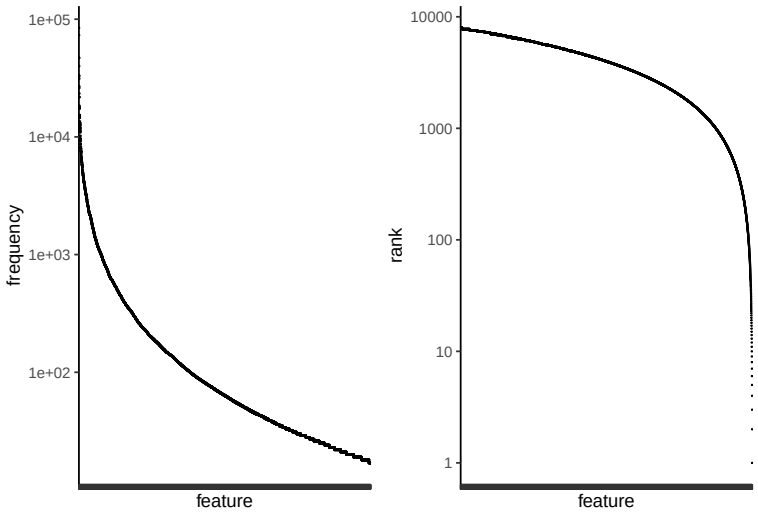
Bisogna utilizzare la funzione *dfm_trim* per eliminare alcune parole del vocabolario con conteggi di basso valore. In tal modo si possono convertire gli oggetti tidy da liste a data.frame o meglio sarebbe utilizzare la libreria *tiytable* per le sue performance (add cite).

```
# clean tokens
com_clean_tidy <- corpus.clean_tidy(com_doge$tidy, mode = "none")
post_clean_tidy <- corpus.clean_tidy(post_doge$tidy, mode = "none")


print("CLEAN COMMENT CORPUS: ")
com_clean_dfm <- com_clean_tidy %>%
```

```
    cast_dfm(document, words, count)
# print(com_clean_dfm)

print("CLEAN POST CORPUS: ")
post_clean_dfm <- post_clean_tidy %>%
    cast_dfm(document, words, count)
# print(post_clean_dfm)
```

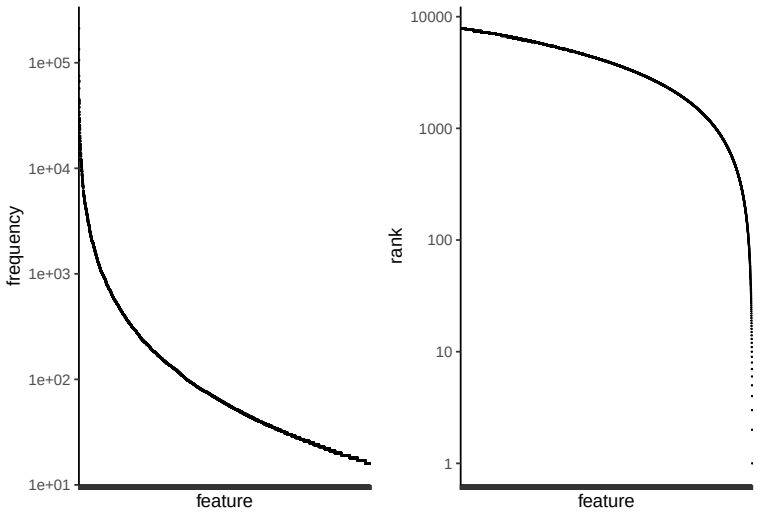
```
print("CLEAN COMMENT CORPUS DIM (docsxtoks): ")
print(dim(com_clean_dfm))
print("CLEAN POST CORPUS DIM: (docsxtoks)")
print(dim(post_clean_dfm))
print("")
print("COLS of TIDY DF")
print(colnames(com_clean_tidy))
print("TYPE OF TIDY DF")
print(typeof(com_clean_tidy))
```

```
freq <- dfm.frequency(com_clean_dfm, 8000)
```




```
head(freq, 10)
```

```
# frequency dfm clean - with stop words
freq <- dfm.frequency(post_clean_dfm, 8000)
```




```
head(freq, 10)
```

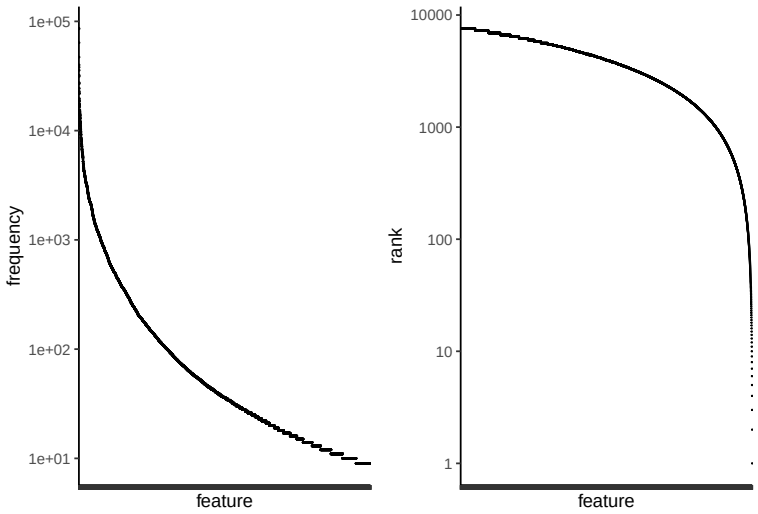
```
# compute tfidf
tfidf <- dfm_tfidf(post_clean_dfm)
topfeatures(tfidf)
```

And, with stemming?

```
com_clean_tidy_stem <- corpus.clean_tidy(com_doge$tidy, mode = "stem")
post_clean_tidy_stem <- corpus.clean_tidy(post_doge$tidy, mode = "stem")


com_clean_dfm_stem <- com_clean_tidy_stem %>%
    cast_dfm(document, words, count)
# print(com_clean_dfm)

post_clean_dfm_stem <- post_clean_tidy_stem %>%
    cast_dfm(document, words, count)
# print(post_clean_dfm)
```

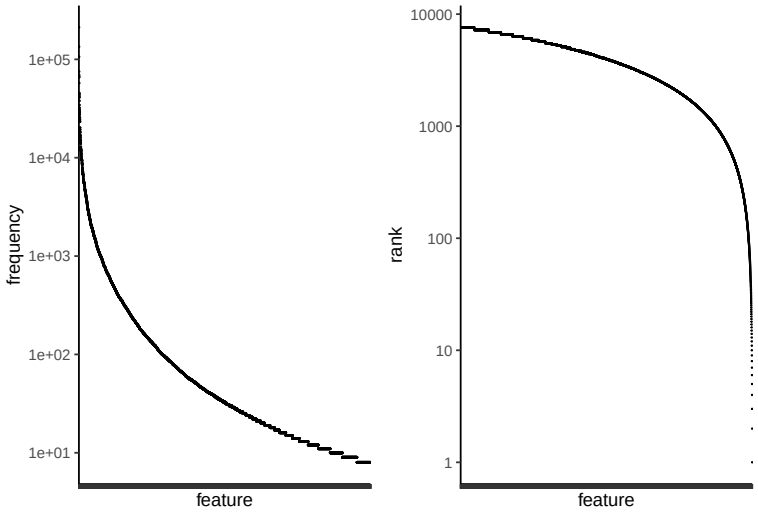
```
print("CLEAN COMMENT CORPUS DIM (docsxtoks): ")
print(dim(com_clean_dfm_stem))
print("CLEAN POST CORPUS DIM: (docsxtoks)")
print(dim(post_clean_dfm_stem))
print("")
print("COLS of TIDY DF")
print(colnames(com_clean_tidy_stem))
print("TYPE OF TIDY DF")
print(typeof(com_clean_tidy_stem))
```

```
# frequency dfm clean - with stemming
freq <- dfm.frequency(com_clean_dfm_stem, 8000)
```




```
head(freq, 10)
```

```
# frequency dfm clean - with stop words - stemming
freq <- dfm.frequency(post_clean_dfm_stem, 8000)
```




```
head(freq, 10)
```

Copiamo i dataframe che ci possono esser utili nell'analisi del dizionario per correggere lessicalmente le parole errate e rimuoviamo tutti gli oggetti che abbiamo utilizzato.

```
tidy_doge_com <- com_clean_tidy
tidy_doge_post <- post_clean_tidy

# remove(data)
remove(com_doge)
remove(post_doge)

remove(com_clean_tidy)
remove(post_clean_tidy)
```

```
remove(com_clean_dfm)
remove(post_clean_dfm)

remove(com_clean_tidy_stem)
remove(post_clean_tidy_stem)

remove(post_clean_dfm_stem)
remove(com_clean_dfm_stem)

remove(freq)
remove(tfidf)
```

Tokenization of Bigrams

```
com_doge <- corpus.tokenize_dfmTidy(data$corpus_comm, ngrams = 2)
print("REDDIT's COMMENT CORPUS: ")
print(com_doge$dfm)
post_doge <- corpus.tokenize_dfmTidy(data$corpus_post, ngrams = 2)
print("REDDIT's POST CORPUS: ")
print(post_doge$dfm)
```

```
com_clean_tidy <- corpus.clean_tidy(com_doge$tidy, ngrams = 2)
post_clean_tidy <- corpus.clean_tidy(post_doge$tidy, ngrams = 2)

print("CLEAN COMMENT CORPUS: ")
com_clean_bigram_dfm <- com_clean_tidy %>%
    cast_dfm(document, words, count)
print(com_clean_bigram_dfm)

print("CLEAN POST CORPUS: ")
post_clean_bigram_dfm <- post_clean_tidy %>%
    cast_dfm(document, words, count)
print(post_clean_bigram_dfm)
```

Tabella di contingenza prima di aver applicato la correzione del dizionario:

DogeCoin Corpus	Type	# Comment	# Post *
#num of docs	Raw	439,054	
	Clean	419,718	
#num of toks	Raw	77,567	
	Clean	59,551	
# hunspell check	TRUE	23,908	22,605
	FALSE	35,643	30,597
#num of stem toks	Clean	44,315	44,329
#num of docs bigrams	Raw	439,054	14,252
	Clean	358,989	14,247
#num of toks bigrams	Raw	1,175,788	1,174,517
	Clean	1,055,811	1,044,181

- with stop words

Analize uncorrect Words Come si è potuto vedere dalla precedente tabella, il numero di parole classificate come lessicalmente non corrette è maggiore della metà del dataset. Si è provato quindi a recuperare almeno le parole maggiormente influenti nel dataset.

Sono state utilizzate le funzioni della libreria *hunspell*, che permette di verificare se una parola risulta corretta o meno, rispetto il loro dizionario , oppure di individuare una lista di suggerimenti per la correzione.

```
# tidy doge comment is compute correct word and stats
correct_com <- tidy_doge_com %>%
    mutate(check = hunspell_check(words))

# aggregate coun for words,docs
correct_post <- tidy_doge_post %>%
    mutate(check = hunspell_check(words))


print(summary(correct_com))
print(summary(correct_post))
```

```
words.spell_checking <- function(data) {
    # counts word
    com_counts <- corpus.countPlot_tidy(correct_post, bool_plot_count = FALSE,
        bool_plot_frequency = FALSE)

    # check words
    com_correct <- com_counts %>%
        mutate.(check = hunspell_check(words))

    print(paste0("RAW WORDS: ", dim(com_correct)[1], sep = " "))

    # aggregate for words
    correct_unique <- aggregate(count ~ words + check, data = com_correct,
        FUN = sum)

    # extract correct
    correct <- correct_unique %>%
        filter(check == TRUE)

    print(paste0("CORRECTS WORDS: ", dim(correct)[1], sep = " "))

    # extract uncorrect
    uncorrect <- com_correct %>%
        filter(check == FALSE)

    print(paste0("UNCORRECTS WORDS: ", dim(uncorrect)[1], sep = " "))

    return(list(correct = correct, uncorrect = uncorrect))
}
```

```
# spell checking add counts and freq.
df <- words.spell_checking(correct_post)
```

```
correct_post <- df$correct
uncorrect_post <- df$uncorrect

remove(df)
```

Una volta estratte le parole riconosciute come corrette e divise da quelle non corrette possiamo procedere l'analisi delle seconde.

```
uncorrect_stem_lem <- uncorrect_post %>%
    cbind(term_stem = uncorrect_post$words) %>%
    cbind(term_lem = uncorrect_post$words) %>%
    word.manipulation(term_stem, mode = 2) %>%
    word.manipulation(term_lem, mode = 1) %>%
    rename(term = words)

dim(uncorrect_stem_lem)
summary(uncorrect_stem_lem)
```

Siamo arrivati a creare un dataframe contente la parola originale estratta dal dataset, la sua lemmatizzazione e stemming. In questo modo nei passi successivi andremo ad aggiungere delle features riguardanti le distanza tra le diverse stringe inserite nel dataframe, così che si possa analizzare la bontà della correzione da parte della funzione di hunspell.

Si è utilizzata la distanza di levenst.....

```
correct_spelling <- function(data, input) {
   data %>%
      mutate( suggest =
             case_when(
               # any manual corrections
               {{input}} == 'aways' ~ 'away',
               {{input}} == 'covid' ~ 'covid',
               {{input}} == 'binance' ~ 'binance',
               {{input}} == 'stonks' ~ 'stonks',
               {{input}} == 'cryptocurrency' ~ 'cryptocurrency',
               {{input}} == 'cryptocurrencies' ~ 'cryptocurrency',
               # check and (if required) correct spelling
               !hunspell_check({{input}}) ~
                  hunspell_suggest({{input}}) %>%
                  # get first suggestion, or NA if suggestions list is empty
                  map(1, .default = NA) %>%
                  unlist() %>%
                  tolower(),
               TRUE ~ {{input}} # if word is correct
            ))
            # if input incorrectly spelled but no suggestions, return input word
           # ifelse(is.na(output), {{input}}, output)
}

# apply suggest
suggest_stem_corFalse <- uncorrect_stem_lem %>%
   correct_spelling(term) %>%
   mutate(dist_term = stringdist::stringdist(term, suggest, "lv")) %>%
```

```
   mutate(dist_stem = stringdist::stringdist(term_stem, suggest, "lv")) %>%
   mutate(dist_source_stem = stringdist::stringdist(term, term_stem, "lv")) %>%
   mutate(dist_lem = stringdist::stringdist(term_lem, suggest, "lv")) %>%
   select(count, term, suggest, term_stem, term_lem, dist_lem, dist_source_stem)

write_rds(suggest_stem_corFalse, "../Data/doge_suggest.rds")
remove(suggest_stem_corFalse)
remove(uncorrect_stem_lem)
remove(uncorrect_post)
remove(correct_post)
```

Abbiamo salvato per precauzione il file essendo che il processo, più volte ripeturo, di calcolo delle distanza può richiedere abbastanza tempo per grandi dataset.

Possiamo adesso effettuare una correzione più raffinata delle parole. Nel prossimo codice ci riferiremo con '*large dictionary*' il dizionario che contiene le parole con i conteggi più altri e con '*medium*' la fascia di mezzo, i loro nomi non rispecchiano quindi le dimensioni del dataset.

La correzione è stata effettuata con i seguenti passi: - flag - if flag term_stem else suggest - for stopwords used suggest - if dist lev from source and stem word is 0 take stem # sigle o parole corte/non modificabili

```
dic_suggest = readRDS("./doge_suggest.rds")


# ????????  holddddd holddddd buyyyy buyyyy
little_correction <- function(input) {
    output <- case_when(input == "hodl" ~ "hold", input == "obis" ~
        "boys", input == "fuckin" ~ "fucking", input == "ios" ~
        "ios", input == "gpu" ~ "gpu", input == "gme" ~ "gme",
        input == "font" ~ "dont", input == "stonks" ~ "stonks",
        grepl("^go*", input) ~ "go", grepl("^usa", input) ~ "usa",
        TRUE ~ input)
    ifelse(is.na(output), input, output)
}

## stop words from snowball
stopwords <- get_stopwords(language = "en", source = "snowball")
stopwords_ <- as.vector(stopwords$word)
remove(stopwords)

# augmented stopwords
other <- purrr::keep(stopwords_, function(x) grepl("['-]+", x))
other1 <- gsub("[[:punct:]]", "", other)
other2 <- unique(gsub("[[:punct:]][a-z]+", "", other))

stopwords <- c(stopwords_, other1, other2)
remove(other, other1, other2)

dim <- dim(dic_suggest)[1]
```

```
# dic with influence uncorret words
dic_l.tdb <- tidytable(dic_suggest) %>%
    filter(count >= 500) %>%
    cbind(flag = -1) %>%
```

```
    cbind(correction = "none") %>%
    select.(count, term, term_stem, term_lem, suggest, flag,
        correction, dist_source_stem)

# flag if a word is in stopwords vec
dic_l.tdb$flag = ifelse.(dic_l.tdb$term_stem %in% stopwords,
    1, 0)

# if is a stopwords, use suggested words
dic_l.tdb$correction = ifelse.(dic_l.tdb$flag == 1, dic_l.tdb$suggest,
    dic_l.tdb$term_stem)

# if lev dist from raw string and stem. string is 0 ..
dic_l.tdb$correction = ifelse.(dic_l.tdb$dist_source_stem ==
    0 & dic_l.tdb$flag != 1, dic_l.tdb$term, dic_l.tdb$suggest)

print("stop words founded")
print(sum(dic_l.tdb$flag))
print("nrow large dictionary")
print(nrow(dic_l.tdb))

print("in %")
print(sum(dic_l.tdb$flag)/nrow(dic_l.tdb) * 100)
```

```
dic_l <- dic_l.tdb %>%
    mutate.(correction = little_correction(correction)) %>%
    select.(count, term, correction, flag) %>%
    mutate.(check = hunspell_check(correction)) %>%
    group_by(flag) %>%
    arrange.(desc(count))

print("Percent of dic_large's correct words over dic_large's dim")
print(nrow(dic_l[check == TRUE]))
print("Percent of dic_large's correct words over total dim")
print(nrow(dic_l[term != correction])/nrow(dic_l) * 100)
```

Si eseguono le medesime operazioni con le parole che hanno una frequenza compresa tra 100 e 500 esclusi. Ci si aspetta di trovare un minor numero di stop words, ma l'obiettivo è tentare di correggere se possibile, se no stemmizzare la parola.

Un problema che non è stato risolto con questa correzione è quello dovuto a caratteri battuti come doppi all'interno delle parole, o comunque risolvere le occorrenze strane all'interno delle parole. Queste verranno escluse dall'analisi del testo, ma non potranno esser utilizzate nei conteggi delle parole, risultando inutili a livello di informazione che portano.

```
# merge with dic of true word
dic_m.tdb <- tidytable(dic_suggest) %>%
    filter.(count > 100 & count < 500) %>%
    cbind(flag = -1) %>%
    cbind(correction = "none") %>%
    select.(count, term, term_stem, term_lem, suggest, flag,
        correction, dist_source_stem)
```

```
# flag if a word is in stopwords vec
dic_m.tdb$flag = ifelse.(dic_m.tdb$term_stem %in% stopwords,
    1, 0)

# if is a stopwords, use suggested words
dic_m.tdb$correction = ifelse.(dic_m.tdb$flag == 1, dic_m.tdb$suggest,
    dic_m.tdb$term_stem)

# if lev dist from raw string and stem. string is 0 ..
dic_m.tdb$correction = ifelse.(dic_m.tdb$dist_source_stem ==
    0 & dic_m.tdb$flag != 1, dic_m.tdb$term, dic_m.tdb$suggest)
print("stop words founded")
print(sum(dic_m.tdb$flag))
print("nrow medium dictionary")
print(nrow(dic_m.tdb))

print("in %")
print(sum(dic_m.tdb$flag)/nrow(dic_m.tdb) * 100)
```

```
dic_m <- dic_m.tdb %>%
    mutate(correction = little_correction(correction)) %>%
    select(count, term, correction, flag) %>%
    mutate.(check = hunspell_check(correction)) %>%
    group_by(flag) %>%
    arrange(desc(count))

head(dic_m, 10)
```

```
dictionary = bind_rows(dic_l %>%
    select.(term, correction), dic_m %>%
    select.(term, correction))

write_rds(dictionary, "../Data/dictionary.rds")
```

```
remove(dictionary)
remove(dic_l)
remove(dic_m)
remove(dic_m.tdb)
remove(dic_l.tdb)
remove(dic_suggest)
```

Remake Corpus

Words manipulation for sentiment

Per scegliere la tecnica da utilizzare tra lemmatizzazione e stemming, riguardo il corpus per eseguire l'analisi del sentimento, si è scelto di verificare il numero di parole escluse dai dizionari una volta eseguito il join con il dataset per i sentimenti.

```
remove(data)
data <- readRDS("../Data/dogecoin.rds")
```

```
lab.dizCompare <- function(.data) {

    raw_data <- corpus.tokenize_dfmTidy(.data, stop = FALSE)
    # tokenize data and take tidy df
    tidy <- raw_data$tidy

    # clean tokens
    corpus_stem <- corpus.clean_tidy(tidy, mode = "stem")
    corpus_lem <- corpus.clean_tidy(tidy, mode = "lemma")
    corpus_raw <- corpus.clean_tidy(tidy, mode = "none")

    # counts tokens
    count_stem <- corpus.countPlot_tidy(corpus_stem, bool_plot_count = FALSE,
        bool_plot_frequency = FALSE)
    count_lem <- corpus.countPlot_tidy(corpus_lem, bool_plot_count = FALSE,
        bool_plot_frequency = FALSE)
    count_raw <- corpus.countPlot_tidy(corpus_raw, bool_plot_count = FALSE,
        bool_plot_frequency = FALSE)

    # build sentiment
    sentiment_stem <- words.computeSentiment(count_stem, plot = FALSE)
    sentiment_lem <- words.computeSentiment(count_lem, plot = FALSE)
    sentiment_raw <- words.computeSentiment(count_raw, plot = FALSE)

    word_raw.source <- as.set(count_raw$words)
    word_lem.source <- as.set(count_lem$words)
    word_stem.source <- as.set(count_stem$words)

    word_lem.sentiment <- as.set(sentiment_lem$words)
    word_raw.sentiment <- as.set(sentiment_raw$words)
    word_stem.sentiment <- as.set(sentiment_stem$words)

    s_raw <- set_cardinality(word_raw.source)
    r_raw <- set_cardinality(word_raw.sentiment)

    s_lem <- set_cardinality(word_lem.source)
    r_lem <- set_cardinality(word_lem.sentiment)

    s_stem <- set_cardinality(word_stem.source)
    r_stem <- set_cardinality(word_stem.sentiment)

    print(paste("Source total words: ", s_raw, "--", "Words used raw x sentiment: ",
        r_raw))
    print(paste("Source total words Lemmatization: ", s_lem,
        "--", "Words used lem x sentiment: ", r_lem))
    print(paste("Source total words Stemming: ", s_stem, "--",
        "Words used stem x sentiment: ", r_stem))

}
```

```
df <- data$corpus_post
lab.dizCompare(df)
```

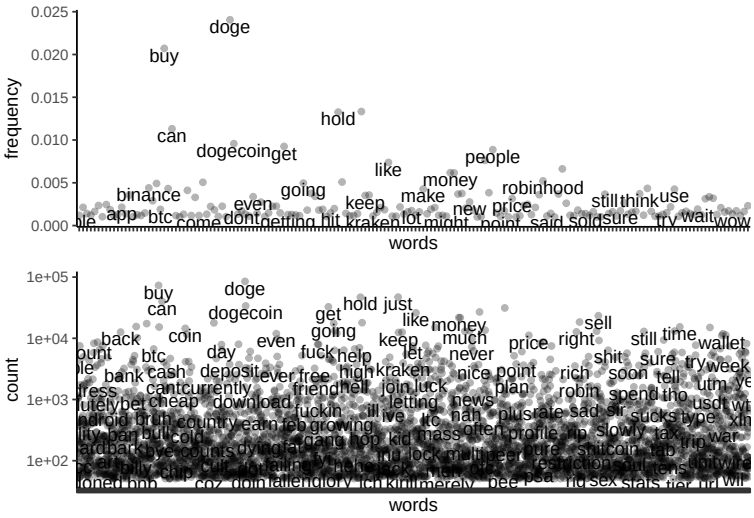
```
remove(df)
remove(data)
```

```
# group with link_id
# aggragate body

doge_txt %>%
   filter(link_id == 't3_ccjlvc')



doge_post_corpus <- corpus_per_post("none")
doge_tidy_post <- create_tidy_corpus(doge_post_corpus)


doge_clean_post <- compute_clean_corpus_tidy(doge_tidy_post$tidy)
# todo:
# remove number issues
doge_post_count <- compute_plot_count_and_freq(doge_clean_post, bool_plot_frequency = FALSE)


head(corpus_doge)

library(stringr)


append(txt_for_post$body[2] )
str_c( txt_for_post$body[2], collapse = ", " , sep = " ")

post <- paste0( txt_for_post$body[2] ,collapse = ", " )
post_string <- paste(post, )
dfm(post)
```

```
# [976] 'aggressiv' [977] 'aggressive' [978] 'aggressively'
# [979] 'aggressiveness' [980] 'aggresssiivvveeellly'
```

Bigrams Analysis

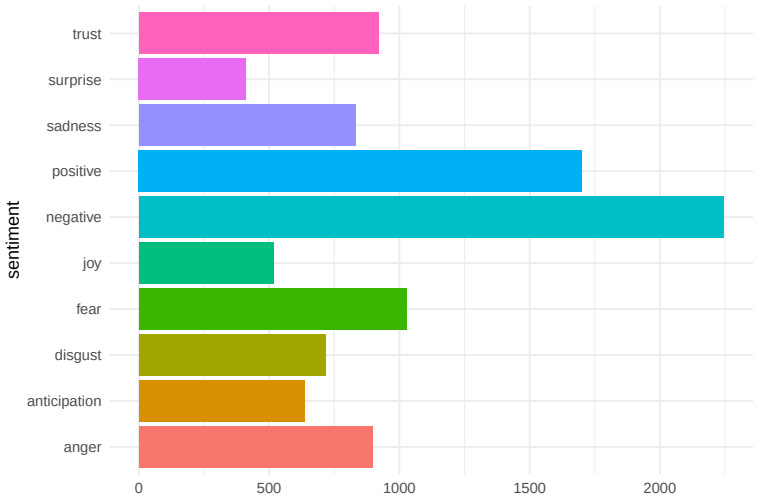
```
data <- readRDS("../Data/dogecoin.rds")
doge_corpus <- data$corpus_comm
doge_tidy <- corpus.tokenize_dfmTidy(doge_corpus)
doge_clean_tidy <- corpus.clean_tidy(doge_tidy$tidy)
doge_word_counts <- corpus.countPlot_tidy(doge_clean_tidy, threshold_count = 50)
```



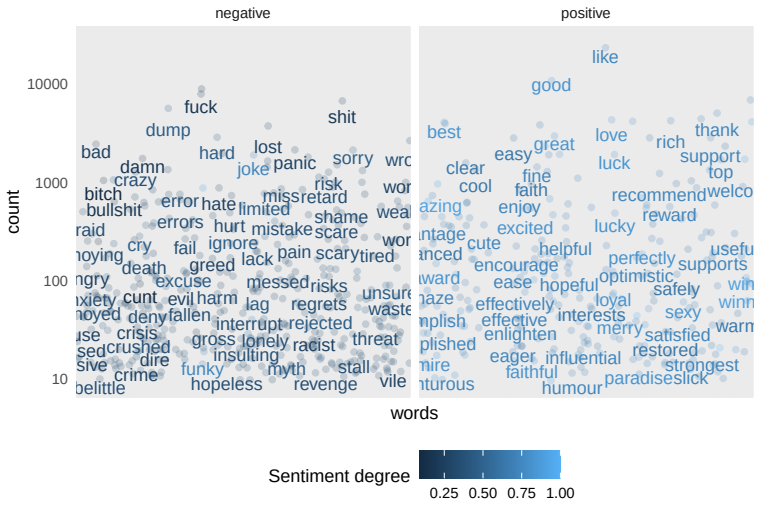

```
set.seed(190)

dff <- doge_tidy$dfm
# colour of words from least to most frequent
textplot_wordcloud(dff, min_count = 5000, color = c("pink", "red",
    "green3", "purple", "orange", "blue3"))
```


```
remove(dff)
```

```
words.classSentiment(doge_word_counts)
```

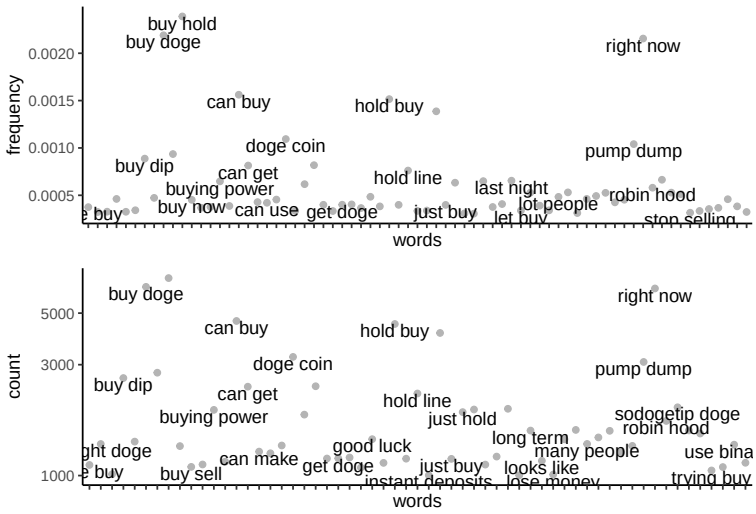



```
doge_word_sentiment <- words.computeSentiment(doge_word_counts,
    n_filter = 10)
```

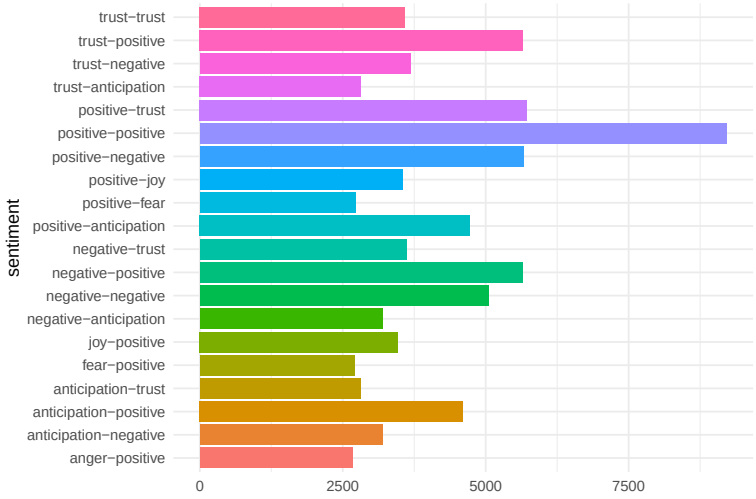



```
remove(doge_tidy)
remove(doge_clean_tidy)
remove(doge_word_counts)
```

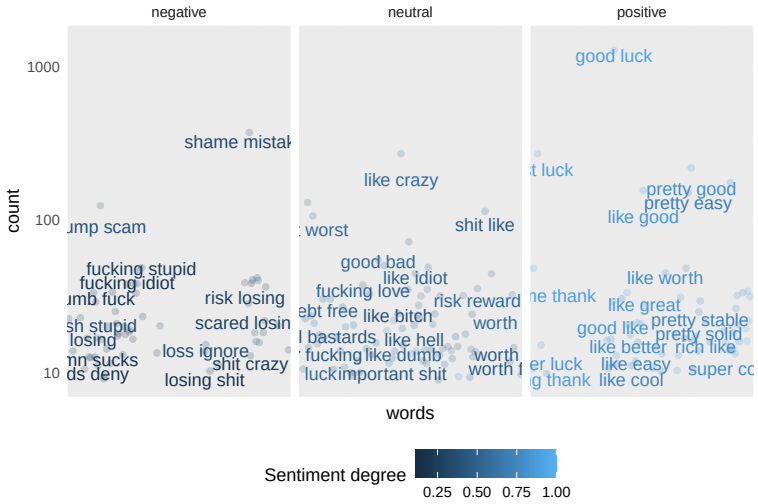
```
doge_tidy_bigrams <- corpus.tokenize_dfmTidy(doge_corpus, ngrams = 2)
doge_clean_tidy_bigrams <- corpus.clean_tidy(doge_tidy_bigrams$tidy,
    ngrams = 2)
doge_bigrams_counts <- corpus.countPlot_tidy(doge_clean_tidy_bigrams,
    threshold_freq = 3e-04, ngrams = 2)
```



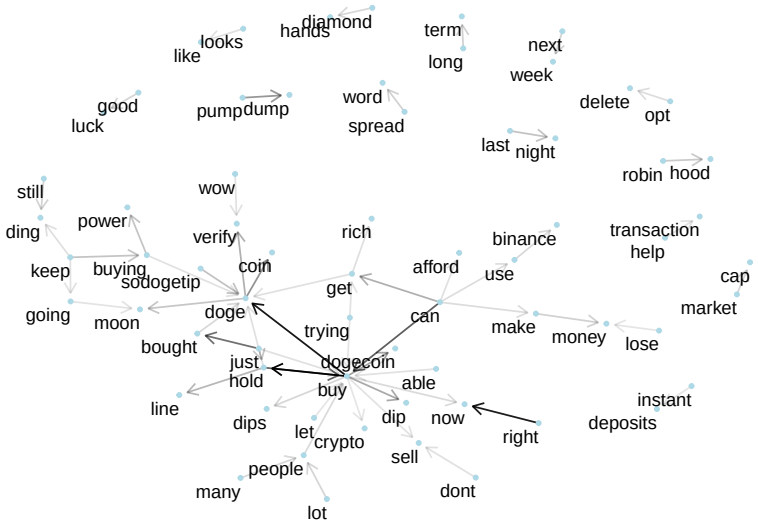

```
bigrams_class_sentiment <- words.classSentiment(doge_bigrams_counts,
    n_filter_sentiment = 2500, ngrams = 2)
```




```
bigrams_sentiment <- words.computeSentiment(doge_bigrams_counts,
    n_filter = 10, ngrams = 2)
```




```
words.network(doge_bigrams_counts)
```




```
remove(doge_tidy_bigrams)
remove(doge_clean_tidy_bigrams)
remove(doge_bigrams_counts)
```

Thurner, Stefan, Rudolf Hanel, Bo Liu, and Bernat Corominas-Murtra. 2015. "Understanding Zipf's Law of Word Frequencies Through Sample-Space Collapse in Sentence Formation." *Journal of The Royal Society Interface* 12 (108): 20150330. https://doi.org/10.1098/rsif.2015.0330.