# Breve introduzione

## Obiettivo del progetto

L'obiettivo di questo progetto è creare dapprima, *from scratch*, un modello di **Sentiment Analysis**; successivamente l'obiettivo è applicarlo ad un caso reale, il tutto costruito attraverso Python3.

## Cos'è un modello di Sentiment Analysis?

Nel campo del machine learning, informalmente, un modello di Sentiment Analysis può essere visto come un insieme di metodi statistici che estrapolano informazioni e soprattutto **sentimenti/opinioni** (delle persone) da un testo libero.

## Quale è il caso reale sul quale applichiamo questo modello?

Una volta costruito il modello, è stato preso un dataset di tweet composto dai tweet di un gran numero di politici americani (assieme alle informazioni sul loro account Twitter) con il fine di analizzarne il testo per poterne ricavare informazioni interessanti.

# Costruzione del modello

## Dataset utilizzato per l'addestramento

### Struttura generale del dataset

Per addestrare e quindi costruire questo modello ho utilizzato un questo dataset (sottoforma di .csv) formato da più di un milione e mezzo di tweet, i quali potevano avere 3 classificazioni (tweet negativo, tweet positivo, tweet neutrale).

### Come il dataset è stato formato

come indicato su questa pagina questo dataset è stato formato (e quindi i tweet che lo compongono) a partire da alcune delle più famose API/servizi di raccolta dati, come *Twitter API*. Ulteriori tecnologie utilizzate (più in generale, non riguardo la raccolta dati) sono sempre annotate al link soprastante.

I tweet non sono stati *labelati* manualmente ma in modo automatico attraverso un algoritmo. Tutto il procedimento per eseguire questo task di labeling è descritto qui. Naturalmente tutte le classificazioni sono state fatte partendo da una definizione di negativo / positivo / neutrale scelta dai creatori di questo dataset (come spiegato nel punto 1.1 del paper soprastante).

### Cosa ho ricavato dal dataset

Di tutto il dataset scaricabile ho mantenuto solamente i campi riguardanti la classificazione (label) e il testo del tweet, oltre ad aver cancellato quei pochi record classificati come neutrali ed aver eliminato il test set per scarsità di tweet (formato solo da circa 500 tweet).

# Preprocessing dei dati

Il Preprocessing dei dati è una attività essenziale e una delle più importanti per andare ad ovviare a problemi intrinseci dei dati e creare un dataset di qualità. I passi che ho provato ad eseguire* per portare a termine questo compito sono, in ordine**, i seguenti:

## fare il *tokenizing* del testo

La tokenizzazione è una tecnica utilizzata per dividere un testo (qualsiasi esso sia) in unità più piccole, come singole parole o termini chiamate token. Per l'occasione, in Python, ho utilizzato il *TweetTokenizer*, cioè un Tokenizer (fornito dal modulo *nltk*) che, in aggiunta alle feature base, lascia intatte la parole con hashtags e menzioni e tratta, rispettivamente il '#' e '@' come token (che vogliamo cancellare), dividendoli così dalla loro parola associata (che invece vogliamo tenere).

esempio: '#casa' –> ['#', 'casa']

## convertire *emoji* ed *emoticons* in testo in lingua inglese

Ho deciso che un insieme (indicato nel file emoji_emoticon.csv) di emoticon ed emoji (scelte personalmente tra quelle più famose/utilizzate) verrà convertito in testo in lingua inglese secondo il seguente criterio: se la emoticon/emoji indica una emozione positiva allora verrà sostiuita con la parola **good**, altrimenti se negativa con **bad**; le emoticon/emoji considerate "neutrali" e tutte quelle non presenti nella lista non vengono valutate e al prossimo passo verranno eliminate.

## eliminare *punteggiatura*, *link*, *url*, *hashtags*, *menzioni* e *stopwords*

eliminiamo tutte quelle informazioni che, con sicurezza, non portano ad una maggior informazione, ma al contrario aggiungono solamente *rumore* (informazioni inutili che potenzialmente peggiorano la qualità dei dati). Una nota particolare va per la punteggiatura: l'apostrofo dovrebbe essere (per poter scrivere alcune parole) mantenuto o eliminato ma trasformando tutte quelle parole (nel caso si possa fare per tutte) che lo utilizzano.

## tentare di correggere le parole *grammaticalmente sbagliate*

Attraverso il modulo *autocorrect* utilizziamo un oggetto di classe *Speller* il cui utilizzo ci permette di, potenzialmente (dato che certe volte a partire da una parola sbagliata è quasi impossibile o molto difficile risalire a quella giusta), correggere una parola sbagliata. Per ragioni di efficienza questo oggetto lavora in modalità "fast", per cui la precisione e la qualità del suo lavoro è più bassa del normale.

## eliminare le parole *non esistenti* nella lingua inglese

Utilizzando la raccolta di tutte le parole in lingua inglese (dato che questo modello opera in inglese) fornito dal modulo *nltk* andiamo ad eliminare tutte quelle parole che non esistono nella raccolta delle parole in lingua inglese; facendo così cioè, andiamo ad eliminare tutte le parole non grammaticalmente corrette e tutti i nomi propri (tranne quelli di persone) di entità più o meno immaginarie/materiali, (pensiamo ai marchi di bevande gassate, automobili, ecc...) che possono essere presenti nei testi dei tweet.

## *lemmatizzare* il testo

Sempre da *nltk* utilizziamo un ogetto di classe *WordNetLemmatizer* che data una parola, ritorna il lemma di quella parola, cioè in modo informale ritorna la forma base "da dizionario" di quella parola (questa forma base cambia da lingua a lingua).

Quindi prenendo la lingua italiana come esempio, se abbiamo due tweet che utilizzano rispettivamente i verbi "corso" e "correrò", tutte e due i verbi saranno ridotti alla parola "correre"

Per migliorare (a livello teorico in modo notevole) la qualità del funzionamento del "lemmatizzatore", si può indicargli il cosiddetto *Pos* (part of speach), cioè di che "tipo" è (un nome, aggettivo, verbo, avvervio, ecc...) la parola che gli stiamo chiedendo di lemmatizzare.

*dato che l'accuratezza risultava peggiore, il passo riguardante, la correzione delle parole grammaticalmente sbagliate, nel modello finale, non viene utilizzato. Sempre per una questione di diminuzione dell'accuratezza viene elminato anche l'apostrofo (per quanto riguarda il passo sulla eliminazione della punteggiatura)

*uno dei criteri che ho seguito per scegliere l'ordine di queste operazioni è stata l'efficienza delle operazioni stesse, operazioni molto pesanti preferibilmente vanno eseguite in un punto in cui gran parte dei dati è stata scremata

# Feature Extraction

Nella branca del machine learning nella quale siamo, con Feature Extraction possiamo indicare tutte quelle tecniche che a partire da un testo hanno l'obiettivo di trasformarlo in un insieme di feature che possono essere utilizzate da un algoritmo. Più nel dettaglio possiamo dire che queste tecniche dato un testo, lo trasformano in una tabella (chiamata **bag of words** o *Bow*) con la quale ogni parola di questo testo (ora diventata una feature) è associata ad un valore numerico. Questa bag of words, alla fine di questo processo non sarà altro che il nostro nuovo dataset

oggi è mercoledi

oggi non piove

se piove prendo un ombrello

| | mercoledi | non | oggi | ombrello | piove | prendo | se | un | è |
|---|---|---|---|---|---|---|---|---|---|
| oggi è mercoledi | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| oggi non piove | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| se piove prendo un ombrello | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

Figure 1: Bow prodotta a partire da 3 frasi con CountVectorizer

## TfidfVectorizer

All'interno di questo progetto ho usato un *Tf-idf Vectorizer*, un algoritmo che trasforma il nostro testo in una bag of word ma con un criterio un po' diverso da un semplice, ad esempio, *Countvectorizer*.

Un TfidfVectorizer produce comunque una bag of words analoga a questa, ma i valori presenti all'interno di essa non sono solo valori dicotomici ma sono generati calcolando il *Tf* (Term frequency) della parola e moltiplicandolo per l'*Idf* (Inverse document frequency) del documento.

Il Term frequency indica quante volte una certa parola è presente in un testo.

L'Inverse Document frequency ed indica l'inverso della *Df* (Document frequency) cioè il numero di documenti dove appare una determinata parola

$$Idf = \log \frac{1}{Df} \quad con\ Df = \#\{testo : testi \mid parola \in testo\}$$

ci si accorge subito, di conseguenza, che il Tfidf è direttamente proporzionale al numero di volte che una parola è presente in una frase ed inversamente proporzionale a quanti documenti quella parola è presente. Una parola con un TfIdf alto è una parola che ha molto più peso rispetto alle altre

### N-grammi

gli n-grammi da definizione sono sottesequenze di una sequenza, che nel nostro caso è il documento; ad esempio: per il documento "oggi è una bella giornata", un trigramma (n-gramma formato da 3 elementi/parole) può essere: "oggi è una", "è una bella", ecc...

Come sappiamo, tutte le parole di tutti i documenti di un dataset vanno a formare gli attributi (le feature, i valori nelle colonne) della nostra bag of words, ma c'è la possibilità di "settare" come feature anche *tutti* gli n-grammi dei documenti di un dataset, con il fine di cercare di aumentare l'accuratezza e la qualità del nostro modello (nel mio modello, i migliori risultati li ho avuti con $n = 3$). Esempio:

è mercoledi

non piove

| | mercoledi | non | è | piove | è mercoledi | non piove |
|---|---|---|---|---|---|---|
| oggi è mercoledi | 1 | 0 | 1 | 0 | 1 | 0 |
| oggi non piove | 0 | 1 | 0 | 1 | 0 | 1 |

Figure 2: Bow esempio utilizzando gli $n$-grammi con $n = 2$

## Feature Selection

Dal nome, la Feature Selection non è altro che quel processo che prevede di selezionare solo una parte più o meno ampia delle features a nostra disposizione. In generale il motivo che porta ad un "taglio" delle features riguarda una questione di efficienza, ma in un numero enorme di feature, soprattutto nel campo ne quale siamo, ci possono essere anche feature "rumorose" che non portano grandi informazioni aggiuntive (quindi pensiamo ad esempio ad una fase di Preprocessing non perfetta).

Nel mio modello ho utilizzato un oggetto di classe SelectKBest, "costruito" passando come parametro *f_classif* e dicendogli (sempre per test empirici) di darmi le 450k feature "più importanti", cioè le 450k feature che hanno ottenuto lo "score" più alto con la funzione di score f_classif (basato sull'ANOVA, Analysis of Variance, ed utilizzabile solo per variabili categoriche)

# Fase di addestramento

## Suddivisione del dataset finale

Per addestrare il mio modello ho deciso di dividere il dataset in tweet di train/test in rispettivamente 87.5/12.5 % dei tweet. Questa percentuale è stata decisa dal fatto che usando all'incirca 1.3 milioni* di tweet, il 12.5 % è comunque un numero molto alto di tweet che possono soddisfare "la domanda di testing".

Facendo varie e continue prove empiriche manuali (non ho utilizzato la cross validation K fold direttamente) con parti diverse del dataset si nota una certa stabilità sui risultati dell'accuratezza.

*Siamo passati da 1.6 milioni a 1.3 milioni di tweet per due motivi: il primo e più importante, 250k tweet sono stati eliminati automaticamente dalla procedura di Preprocessing, che eliminando e pulendo il tweet dalle parti inutili, dobbiamo presumere che li abbia eliminati completamente; altri 30k tweet vengono tolti per bilanciare le classi.

## Utilizzo degli algoritmi e i loro risultati

Gli algoritmi forniti da sklearn (e le loro accuratezze) che ho provato ad utilizzare per l'addestramento sono i seguenti:

| algoritmo | accuratezza |
|---|---|
| BernoulliNB | 82.74 % |
| MultinomialNB | 78.54 % |
| ComplementNB | 78.53 % |
| LinearSVC | 78.34 % |
| LogisticRegression | 77.12 % |
| RandomForestClassifier | 73.53 % |
| KNeighborsClassifier | 66.56 % |
| DecisionTreeClassifier | 58.41 % |
| SGDClassifier | 56.54 % |

**In generale, come previsto, algoritmi come Naive Bayes (e quindi tutte le varianti) hanno una buona accuratezza.

**non sono state fatte *Grid* o *Random Search* dato che in qualsiasi dei due casi bisognerebbe aver avuto una conoscenza non banale, in ogni modo, degli *iperparametri*

# Applicazione del modello per il caso reale

Come accennato in precedenza nell'introduzione, una volta aver creato il modello lo utilizziamo per applicarlo al caso reale descritto.

## Dataset utilizzato

### Struttura

Il dataset utilizzato è stato trovato su Reddit ed è composto da due file .json:

- il primo (original_tweets.json) è composto da 1.25 milioni di tweet sottoforma di oggetti JSON (presi da una API per accedere ai contenuti di Twitter, come ad esempio la stessa Tweepy). La forma dei tweet in JSON è spiegata chiaramente qui
- il secondo (original_users.json) è composto dagli oggetti JSON dei profili Twitter dei politici che hanno scritto i tweet indicati precedentemente. La forma dei tweet in JSON è spiegata chiaramente qui

come indicato nella pagina dove è stato recuperato il dataset, i tweet sono stati recuperati il 6 giugno del 2017. Per cui tutti i ragionamenti da effettuare sui Tweet sono da fare in funzione di quella data (ad esempio l'account @POTUS, famoso per essere del President Of The United States era l'account di Donald Trump).

### creazione del Dataset

Naturalmente non tutte le informazioni fornite dagli oggetti JSON (sia per quanto riguarda il file con i tweet, sia il file con le informazioni degli utenti di quei tweet, cioè i politici) sono utili o interessano per creare il dataset.

per la parte del dataset dedicata ai tweet dei politici
(file political_analysis/preprocess_data/tweets.csv), di tutti i campi presenti, abbiamo tenuto i seguenti campi, in modo tale che i campi del file .csv finale si presentino in questo modo:

- politician_name
- text
- original_text_lang
- created_at

per la parte del dataset dedicata agli account dei politici
(file political_analysis/make_analysis/users.csv), di tutti i campi presenti, abbiamo tenuto alcuni campi e aggiunti dei nuovi, in modo tale che i campi del file .csv finale si presentino in questo modo:

- politician_name
- account_location
- followers_count
- posted_tweet_number
- account_created_at

Da notare, per quanto riguarda il file con gli account degli utenti, che esiste un campo (che non è presente nei campi dell'oggetto JSON dell'utente) denominato *political_party*. Dal nome, appunto, è il partito politico di appartenenza. La procedura per ovviare (in parte) alla mancanza delle informazioni sui partiti politici è presente nella cartella /select_data/set_political_party/ . Sono state recuperate da github ( qui, qui e qui) delle liste con informazioni sui politici americani, tra le quali erano presenti anche i loro partiti di appartenenza. Questa procedura ha ovviato al problema per circa 80% dei politici; il resto dei politici rimasti "senza partito" lo hanno ricevuto "a mano".

Successivamente per quanto riguarda il campo *account_location*, desiderando di avere solamente al suo interno il nome dello stato (ad esempio California, Texas, ecc..) invece ho provveduto ad aggiungere un piccolo pezzo di codice all'interno del file /political_analysis/select_data/load_users.py) che, come nel caso precedente, ha tamponato in buona parte questo problema: la maggior parte degli utenti aveva come account_location NomeCittà, NomeContea, codicePostaleStato (ad esempio, Los Angeles, XYZCounty, CA) mentre interessava solamente, in questo caso, il nome California.

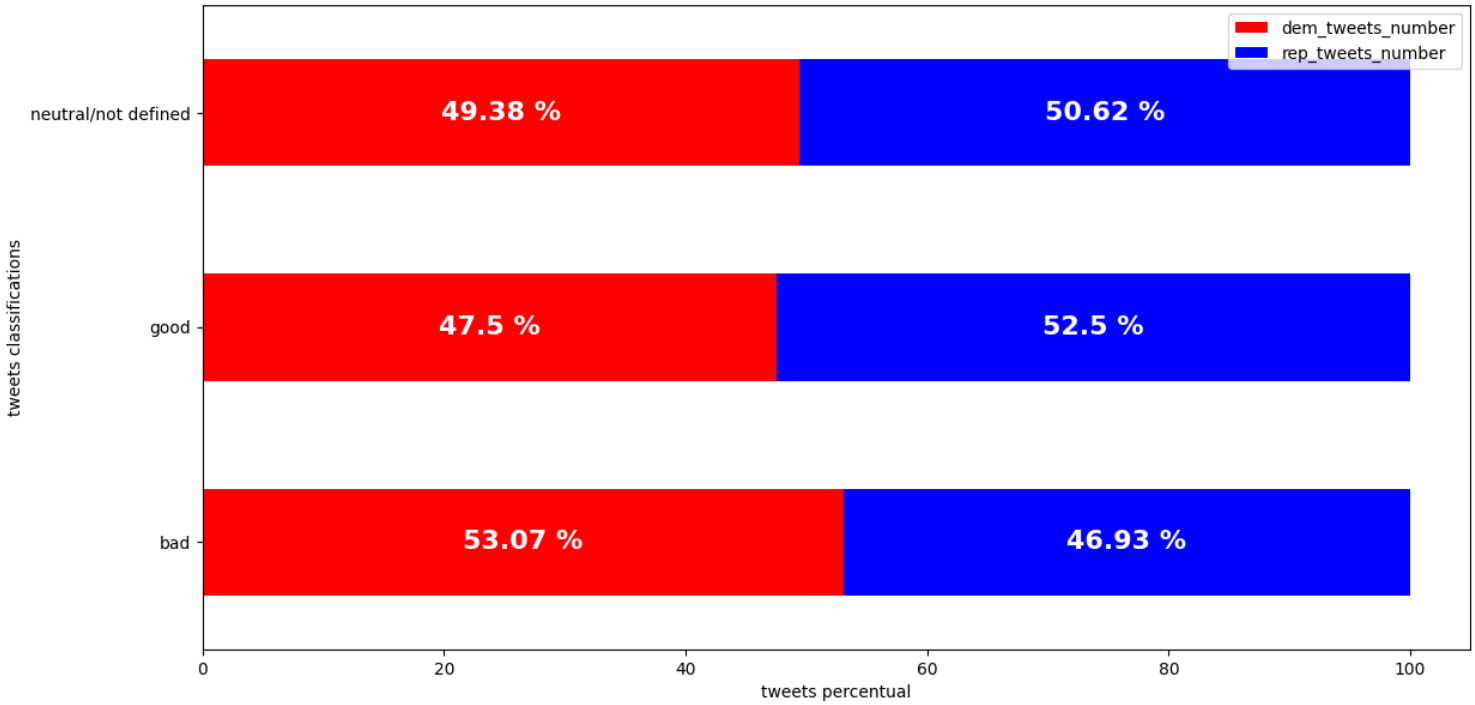
# analisi dei risultati

Ho deciso di utilizzare parte dei risultati e delle informazioni ricavate dal labeling dei dati per produrre alcuni grafici e analizzare i risultati.

## grafici sulla relazione partiti politici - classificazione tweet

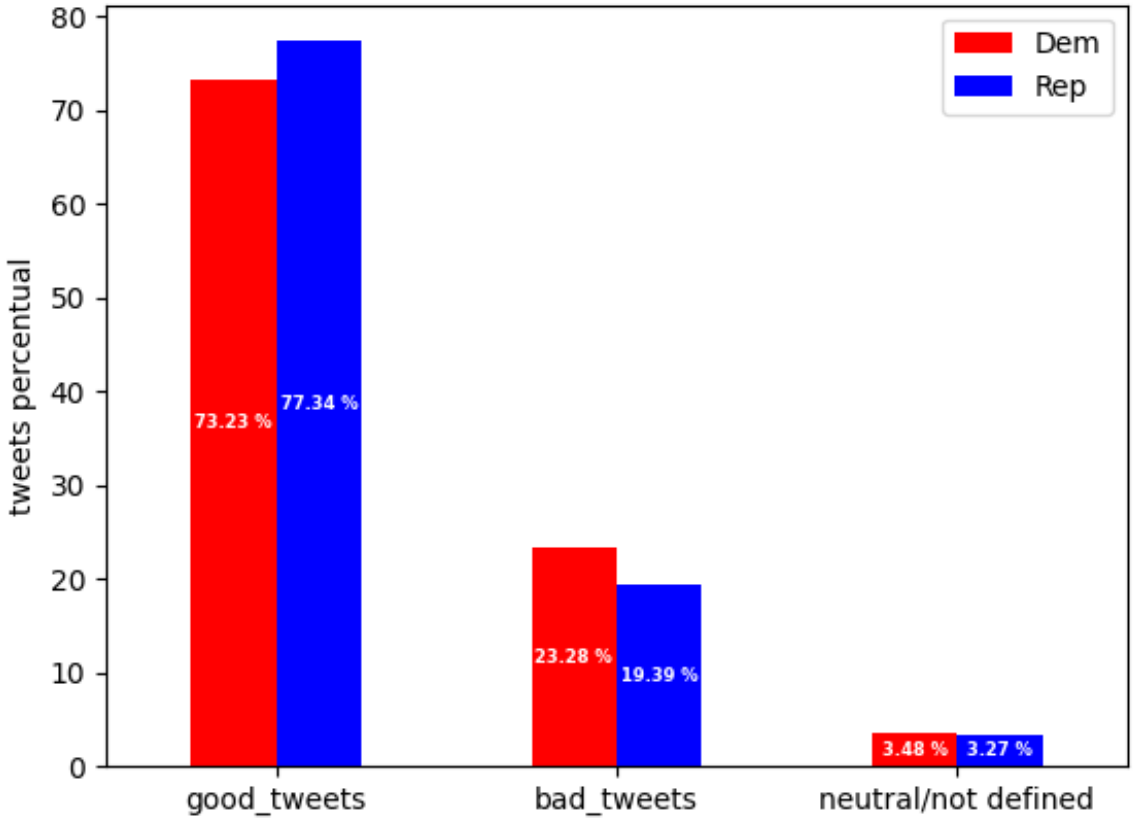
### primo grafico

ho sommato tutti i tweet buoni - cattivi e neutrali/non definiti e per ognuna di queste somme ho calcolato in percentuale quanti tweet appartenessero a un partito rispetto ad un altro



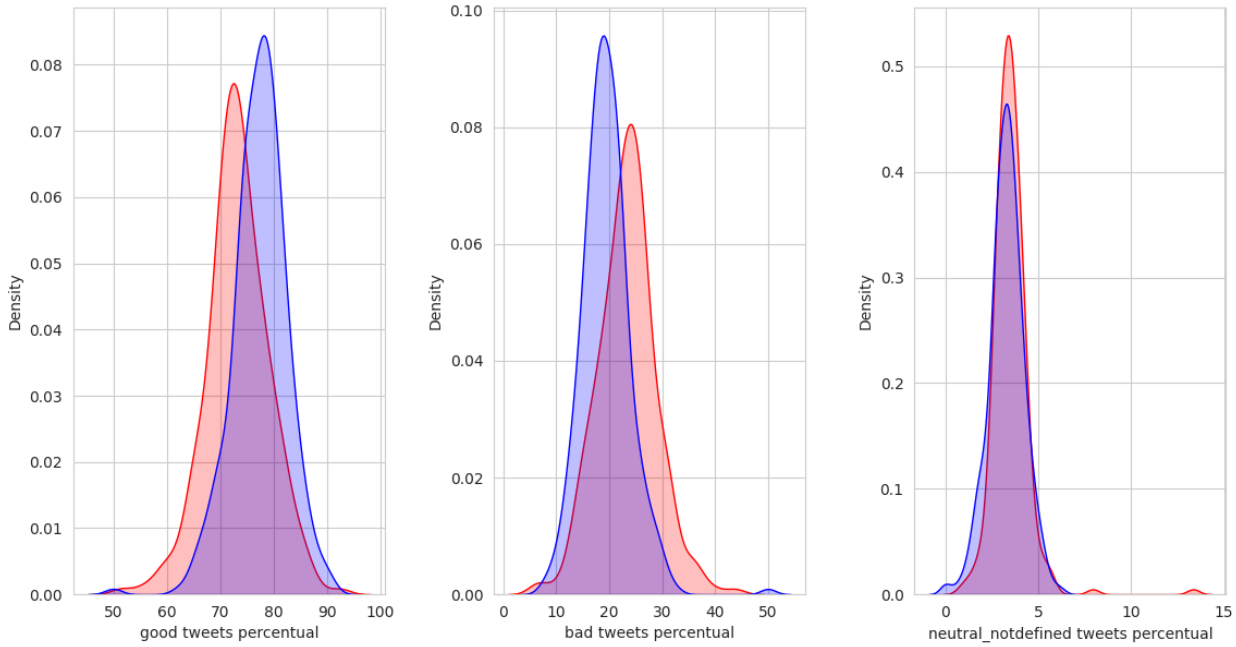
**secondo grafico**

ho calcolato per ogni politico quanti tweet buoni, cattivi e neutrali/non definiti ha pubblicato e ho calcolato le percentuali sui suoi tweet totali. successivamente per ogni partito ho calcolato la media dei tweet buoni, cattivi e neutrali/non definiti dei propri politici



**terzo grafico**

ho calcolato per ogni politico quanti tweet buoni, cattivi e neutrali/non definiti ha pubblicato 'successivamente in ogni subplot ho plottato le distribuzioni dei tweet buoni, cattivi e neutrali/non definiti dei membri di ogni partito



## grafico sulla relazione numero di follower - classificazione tweet

Con questi grafici viene evidenziata la "distribuzione" delle varie categorie di classificazione dei tweet in base al numero dei follwers (semplicemente ho preso il numero di followers di tutti i politici con la loro "distribuzione" di tweet positivi, negativi, ecc.. e ho plottato tutte le coppie/punto)

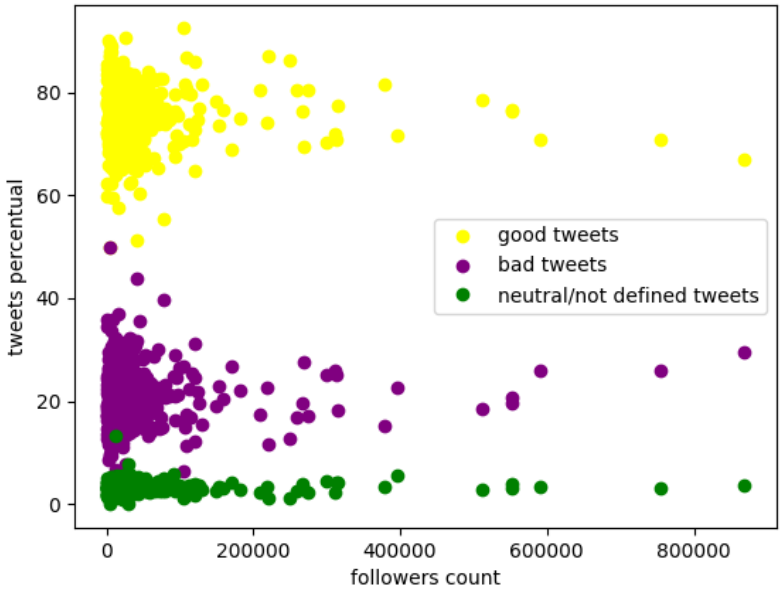



Figure 3: grafico a partire da 0 fino a 900k followers

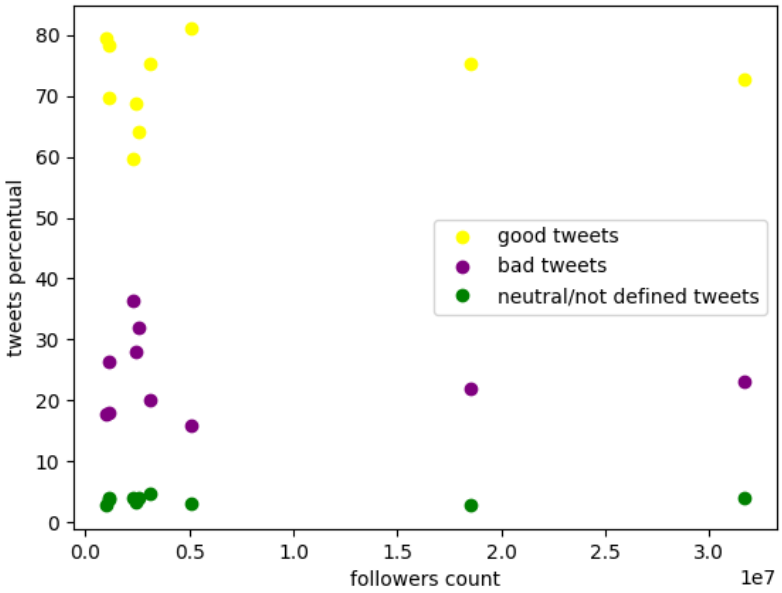



Figure 4: grafico da 900k followers

## grafici sulla relazione stati - classificazione tweet

### primo grafico

ho sommato tutti i tweet buoni, cattivi e neutrali/non definiti dei politici di uno stato (indipendentemente che siano democratici o repebblicani) e ho rappresentato qui sul grafico le percentuali dei tweet buoni per ogni stato sui tweet totali.

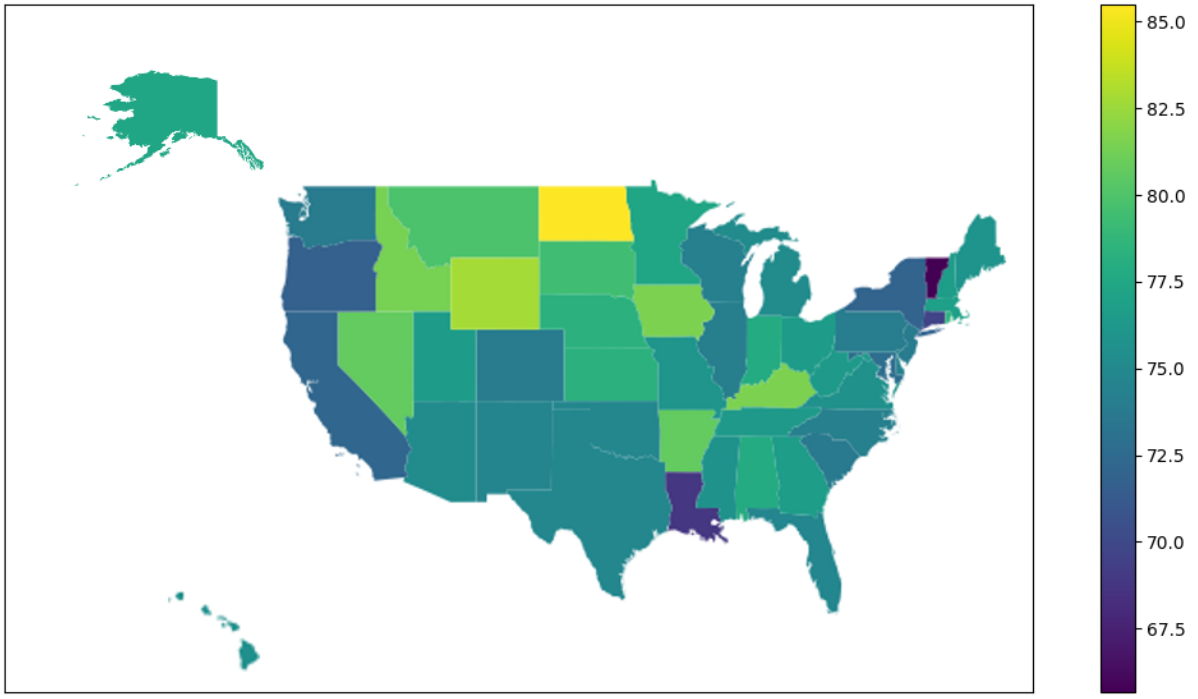




Figure 5: mappa per riconoscere la locazione degli stati

**secondo grafico**

ho calcolato per ogni politico quanti tweet buoni, cattivi e neutrali/non definiti ha pubblicato e ho calcolato le percentuali sui suoi tweet totali. Successivamente per ogni stato ho calcolato la media dei tweet buoni, cattivi e neutrali/non definiti dei propri politici e qui nel grafico ho rappresentato le percentuali dei tweet buoni sui tweet totali.

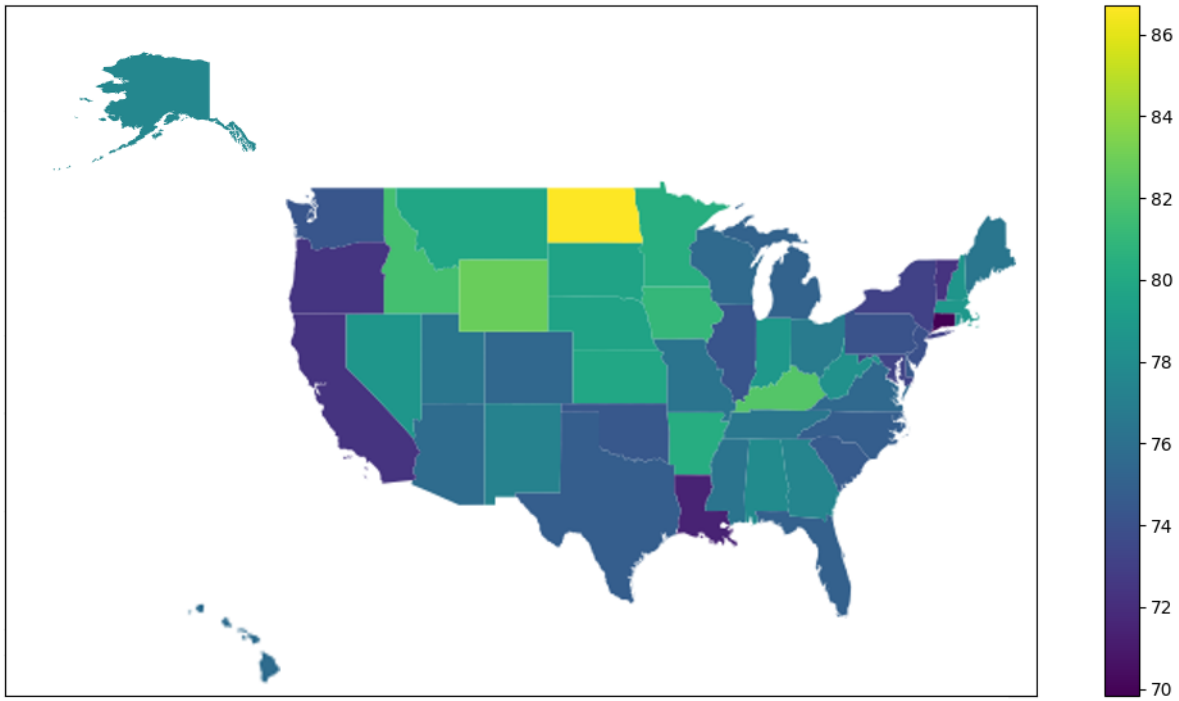




Figure 6: mappa per riconoscere la locazione degli stati

# Moduli di Python 3.6.9 utilizzati

- scikit-learn 0.23.1
- nltk 3.5
- pandas 1.0.4
- geopandas 0.9.0
- seaborn 0.11.1
- matplotlib 3.2.1
- numpy 1.19.5
- joblib 1.0.0
- autocorrect 2.3.0