Chi lo ha detto?

Uso della classificazione per riconoscere l'autore di un tweet

Marco Cavalli - X81000445

A.A. 2019-2020

Sommario

Introduzione

Il mondo odierno è sempre più interconnesso. Molto di quello che riusciamo a fare grazie alle nuove tecnologie in ambito informatico era impensabile anche fino a pochi decenni fa.

Un aspetto centrale della rivoluzione informatica degli ultimi anni è stata la presenza dei social network: questi mezzi hanno cambiato totalmente il modo in cui interagiamo tra noi.

Sebbene queste tecnologie abbiano apportato dei netti miglioramenti al nostro stile di vita, hanno però introdotto nuovi problemi di una certa entità come le **fake news,** come per esempio quelle generate dai **fake accounts**.

Un metodo per poter risolvere il problema di riconoscere i fake accounts è quello di usare algoritmi di **Machine Learning**.

Lo scopo di questo progetto è quello di introdurre dei metodi di Machine Learning basati su classificazione, che permettano il riconoscimento dell'autore di un tweet. Si darà inoltre anche una spiegazione su come questo metodo possa essere una maniera (magari non la migliore) per rispondere a questo problema.

Obiettivi

Date due (o più) persone e un insieme di tweet cercheremo di stabilire chi dei due sia l'autore di ciascun tweet presente nell'insieme. Per far questo dovremo:

- Fornire un metodo che crei un insieme di Tweet usando le API di Twitter
- Pre-processare i tweet usando NLP
- Fornire dei classificatori
- Classificare nuovi tweet in base ai dati già conosciuti

Contenuto della relazione

Mostreremo un primo esempio usando una lista di tweet di Trump e Hillary Clinton così da spiegare cosa fa il codice e come si comporta la classificazione basata su regressione logistica.

Successivamente mostreremo come funziona lo script in caso ci siano più di due etichette (aggiungeremo i tweet di Barack Obama).

Ci concentreremo anche nel valutare le prestazioni dei classificatori basati su regressione logistica, KNN e Naive Bayes Multinomiale sia con 2 che con 3 etichette.

Sarà anche descritta un'applicazione web, che fa un uso pratico delle tecniche di classificazione discusse, con annessa guida di utilizzo.

Infine, parleremo del codice nello specifico, descrivendo i vari metodi presenti nel codice, elencando i parametri e fornendo una spiegazione sul compito che svolgono.

Promemoria

Useremo le API di Twitter per ricavare i dati su cui fare i nostri studi. Twitter ha imposto nel tempo dei **limiti** sull'utilizzo di queste API.[1]

Difficilmente usando questo software si supereranno questi limiti (molti dei quali legati ad aspetti dell'esperienza consumer su Twitter che non vengono affrontati nemmeno, come lo scrivere un Tweet e pubblicarlo), ma è importante essere consapevoli che un uso **improprio** o **smodato** delle API di Twitter potrebbe comportare una **limitazione** (se non un **blocco**) sull'uso delle API stesse.

La porzione di codice che gestirà l'accesso alle API sarà simile a questa:

```
import twitter

ck = input("consumer_key: ")
cs = input("consumer_secret: ")
atk = input("access_token_key: ")
ats = input("access_token_secret: ")

twitter_keys = {
    'consumer_key':        ck,
    'consumer_secret':     cs,
    'access_token_key':    atk,
    'access_token_secret': ats
}

api = twitter.Api(
    consumer_key        =   twitter_keys['consumer_key'],
    consumer_secret     =   twitter_keys['consumer_secret'],
    access_token_key    =   twitter_keys['access_token_key'],
    access_token_secret =   twitter_keys['access_token_secret'],
    tweet_mode = 'extended'
)
```

Fondamentale usare l'autenticazione perché la versione base delle API non ci permetterebbe di fare molto.

Per poter ricavare i quattro codici richiesti bisogna richiedere l'accesso come sviluppatori su https://developer.twitter.com/.

Una volta ottenuto l'autorizzazione da Twitter per sviluppare app usando le sue API si potranno usare queste ultime per ricavare i tweet utili a formare il nostro dataset.

[1] Regolamento ufficiale all'indirizzo: https://developer.twitter.com/en/docs/basics/rate-limiting

Classe TweetMiner e mining dei tweet

Per facilitare il nostro lavoro e poter serializzare lo scraping di considerevoli quantità di tweet da Twitter, useremo una classe che ha lo scopo di fornire i metodi specifici. La classe si chiama **TweetMiner**. Questa classe contiene un metodo chiamato `mine_user_tweets` che non fa altro che ritornare una lista di tweet di un utente ben specifico (specificato con il parametro "user"). È possibile indicare un numero massimo di tweet da prelevare (parametro "max_tweets") e un numero di tentativi massimo per evitare loop infiniti se l'utente ha meno tweet del previsto (parametro "max_pages"). Per maggiori informazioni sugli altri parametri consigliamo di attenzionare i paragrafi finale di questa relazione dove viene trattato in dettaglio il codice.

```
class TweetMiner(object)
def __init__(self, api, result_limit = 100, max_tweets=3000)
def mine_user_tweets(self, user="", mine_retweets=False, no_replies=True,
max_pages=40)
```

Istanziato un oggetto della classe TweetMiner, si può procedere con il mining dei tweet. Per fare ciò basta fornire alla nostra istanza il nome dell'utente da cui prelevare tweet. Per evitare di reiterare questa operazione ogni volta che dobbiamo inizializzare il nostro dataset, è previsto un metodo che permette di salvare i dati in memoria in una cartella denominata "**DATA**" con nome lo stesso della **username** e in formato **csv**.

```
trump = 'realDonaldTrump'

try:
    trump_df = pd.read_csv('../Data/{}.csv'.format(trump))
    notExists = False
except:
    notExists = True

ans = ''

if notExists == False:
    while True:
        ans = input("Trovato un file contenente i tweet dell'utente
cercato. Vuoi sovrascrivere il contenuto di questo file? [s/n] ")
        if(ans == 's' or ans == 'n'):
            break

if ans == 's' or notExists == True:
    trump_tweets = miner.mine_user_tweets(user=trump)
    trump_df = pd.DataFrame(trump_tweets)
    trump_df.to_csv('../Data/{}.csv'.format(trump))
    print("Dati sovrascritti nel percorso '../Data/{}.csv!'".format(trump))
else:
    print("Useremo i dati contenuti nel file salvato precedentemente!")
```

Questo estratto di codice ci permette di notare come il software si comporta quando chiediamo di aggiungere i tweet di un nuovo user. Viene prima controllato se non esiste già un file salvato di quel particolare username. Se esiste viene chiesto se si vuole sovrascrivere e quindi continuare con il mining, altrimenti si possono riutilizzare i dati già presenti in quel file.

Creazione del dataset

Una volta definito come prelevare i dati da Twitter si può procedere con la formazione del dataset. Nella fattispecie ci limiteremo ad un caso con due etichette (Donald Trump e Hillary Clinton). Successivamente verrà trattato anche un caso con tre etichette.

```
hillary_df = pd.read_csv('../Data/{}.csv'.format(hillary))
trump_df = pd.read_csv('../Data/{}.csv'.format(trump))
dataset = pd.concat([hillary_df, trump_df], axis=0)
dataset = dataset.sort_values(by='retweet_count')
dataset = dataset.reset_index(drop=True)
```

	tweet_id	handle	retweet_count	text	mined_at	created_at
0	1077	realDonaldTrump	2	I don't know Putin have no deals in Russia and...	2020-02-22 01:07:58.785460	02-07-2017 12:04:01
1	1110610159188041729	HillaryClinton	9	@mintimm @kenscudder Congratulations x2, Mini!...	2020-02-21 17:05:23.728000	Tue Mar 26 18:31:10 +0000 2019
2	1067	realDonaldTrump	11	I hearby demand a second investigation after S...	2020-02-22 01:07:58.785460	03-03-2017 20:49:53
3	1133369001617350656	HillaryClinton	11	@Hasbro Thank you. 🙏	2020-02-21 17:05:23.299601	Tue May 28 13:46:41 +0000 2019
4	1067988841431277569	HillaryClinton	15	@tarabea80 @BillClinton What kind words Tara. ...	2020-02-21 17:05:24.074316	Thu Nov 29 03:49:16 +0000 2018
...	...	...	...	...	...	...
4957	1007376361101582336	HillaryClinton	193633	But my emails. https://t.co/G7TIWDEG0p	2020-02-21 17:05:24.849021	Thu Jun 14 21:36:55 +0000 2018
4958	1199718185865535490	realDonaldTrump	194251	https://t.co/11nzKwOCtU	2020-02-21 18:10:59.093486	Wed Nov 27 15:54:39 +0000 2019
4959	829846842150096896	HillaryClinton	219247	3-0	2020-02-21 17:05:25.690788	Fri Feb 10 00:17:59 +0000 2017
4960	870090101765931008	HillaryClinton	281317	People in covfefe houses shouldn't throw covfe...	2020-02-21 17:05:25.690788	Thu Jun 01 01:30:20 +0000 2017
4961	796394920051441664	HillaryClinton	649760	"To all the little girls watching...never doub...	2020-02-21 17:05:25.690788	Wed Nov 09 16:51:59 +0000 2016

4962 rows × 6 columns

Il codice mostrato fa semplicemente questo:

- In un momento successivo al mining dei tweet, preleva in memoria i dati degli utenti forniti in input e li salva in DataFrame dedicati
- Concatena i due DataFrame
- Riordina le righe per numero di retweet (motivato per l'intenzione di dare una prima mescolatura dei dati in vista dello splitting)
- Resetta l'indice

Avremo così creato il dataset finale dei tweet di Donald Trump e Hillary Clinton.

Nel sorgente completo sono previsti anche metodi per la gestione dell'aggiunta di nuovi utenti al dataset e per la rimozione degli stessi. Per semplicità, si rimanda sempre alla sezione del codice per informazioni più specifiche al riguardo.

Preprocessing dei tweet

Prima di trattare la problematica della classificazione è fondamentale fornire una funzione di rappresentazione per permettere ai classificatori di poter trattare i dati (che nel nostro caso sono dei tweet).

Per fare ciò utilizzeremo un metodo di pre-processing chiamato **cleanTweet** e due classi usate per la vettorizzazione (**CountVectorizer**) e la normalizzazione (**TfidfTransformer**). Riprenderemo le due classi tra qualche paragrafo; per ora ci concentreremo sul metodo cleanTweet.

```
tweet_text = dataset['text'].values
clean_text = cleanTweet(tweet_text)

import textacy.preprocessing as txt

def cleanTweet(raw_data):
    data = [txt.replace_urls(x,"") for x in raw_data]
    data = [txt.replace_emails(x,"") for x in data]
    data = [txt.replace_emojis(x,"") for x in data]
    data = [txt.replace_user_handles(x,"") for x in data]
    data = [txt.replace_phone_numbers(x,"") for x in data]
    data = [txt.replace_numbers(x,"") for x in data]
    data = [txt.replace_currency_symbols(x,"") for x in data]
    data = [txt.replace_hashtags(x,"") for x in data]
    return data
```

`cleanTweet` non fa altro che pre-processare i dati che gli si forniscono (nel nostro caso sono i messaggi contenuti nei tweet) eliminando in serie:

- URL
- Email
- Emoji
- Username
- Numeri di telefono
- Numeri
- Simboli di moneta
- Hashtag

Segue un esempio di funzionamento del pre-processing sui dati del dataset.

```
["I don't know Putin have no deals in Russia and the haters are going crazy - yet Obama can make a deal with Iran #1in terror n
o problem!"
 '@mintimm @kenscudder Congratulations x2, Mini! Such wonderful news.'
 'I hearby demand a second investigation after Schumer of Pelosi for her close ties to Russia and lying about it. https://t.co/
qCDljfF3wN'
 '@Hasbro Thank you. 🙏'
 '@tarabea80 @BillClinton What kind words Tara. It was great meeting you last night, and thanks for writing!'
 '@KatieS @MoxxieVentures This is exciting, Katie. Congratulations—and love the name!'
 'I hear by demand a second investigation after Schumer of Pelosi for her close ties to Russia and lying about it. https://t.c
o/qCDljfF3wN'
 '@itsmebeccam @TATLGDoc I’m so proud of you Rebecca. And you should be proud of all you’ve accomplished. There’s a bright futu
re ahead!'
 "@KLSecondCity Please pass on to Nora that I'm thinking of her, rooting for her, and a proud member of #TeamNora."]
```

Quando li preleviamo, i Tweet mantengono esattamente il loro formato "ufficiale". Allo stato attuale non sono per nulla utilizzabili: contengono tanti caratteri speciali come le emoji, e diversi costrutti come e-mail e hashtags che non ci danno informazione alcuna sull'autore. La scrematura iniziale che fa il metodo cleanTweet porta gli stessi Tweet allo stato seguente:

```
["I don't know Putin have no deals in Russia and the haters are going crazy - yet Obama can make a deal with Iran #1in terror n
o problem!", '  Congratulations x2, Mini! Such wonderful news.', 'I hearby demand a second investigation after Schumer of Pelos
i for her close ties to Russia and lying about it. ', ' Thank you. ', '  What kind words Tara. It was great meeting you last ni
ght, and thanks for writing!', '  This is exciting, Katie. Congratulations—and love the name!', 'I hear by demand a second inve
stigation after Schumer of Pelosi for her close ties to Russia and lying about it. ', '  I'm so proud of you Rebecca. And you s
hould be proud of all you've accomplished. There's a bright future ahead!', " Please pass on to Nora that I'm thinking of her,
rooting for her, and a proud member of ."]
```

Questo primo pre-processing ci permetterà di avere dei dati in un formato più semplice da trattare e permetterà di rappresentarli eliminando termini inutili.

Splitting del dataset

Lo splitting del dataset in due parti (training e test set) è fondamentale per poter valutare correttamente l'operato del nostro codice. Lo effettueremo usando **train_test_split** del modulo **sklearn.model_selection**. La porzione di codice che si occupa dello split è la seguente:

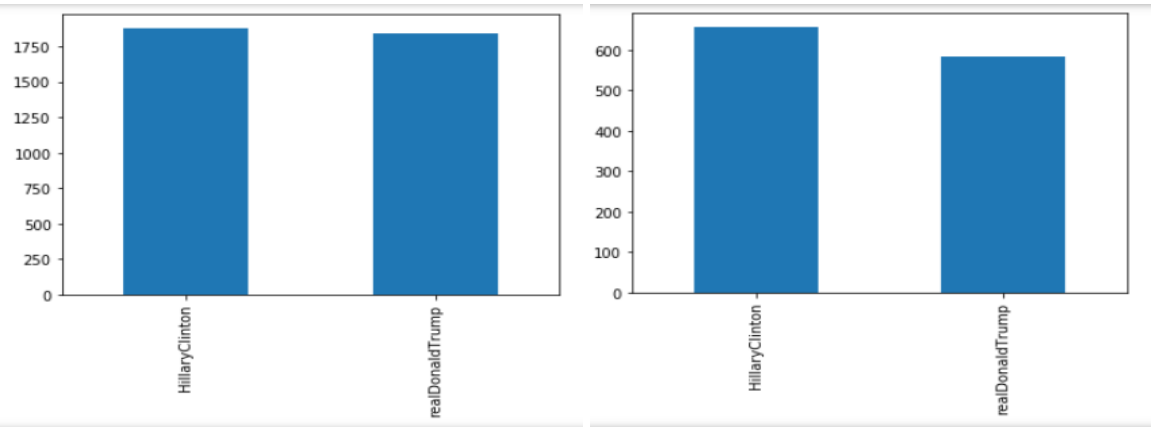
```
from sklearn.model_selection import train_test_split
import numpy as np
from matplotlib import pyplot as plt

np.random.seed(1234)
train_set, test_set, y_train_set, y_test_set  =
train_test_split(clean_df['x'].tolist(),clean_df['label'].tolist(),
test_size=0.25)
```

Fissato un seed (decisione presa per poter riottenere lo stesso split ogni volta che ci serve in questa fase dimostrativa), **train_test_split** divide il dataset di tweet (ora chiamato clean_df in quanto è stato pre-processato) in due parti e torna quattro valori:

1. train_set e y_train_set che sono i due set del training set che contengono i tweet e i rispettivi autori
2. test_set e y_test_set che sono I due set del test set che contengono i tweet e i rispettivi autori

I due set saranno così composti:



I set sono abbastanza equilibrati (in particolare il training set). E' fondamentale che i set siano abbastanza grandi ed equilibrati: anche dalla loro composizione dipenderanno le prestazioni del classificatore.

Verso la classificazione

Prima di effettuare la classificazione, dobbiamo necessariamente trasformare tutti i nostri tweet in un formato adatto alle esigenze del classificatore. Come abbiamo accennato in un paragrafo precedente, dovremo rappresentare i dati sottoforma di vettori normalizzati. Per fare ciò, useremo le classi **CountVectorizer** e **TfidfTransformer**.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

count_vect = CountVectorizer(ngram_range=(2,4), max_features=3000,
max_df=0.8, strip_accents='unicode', stop_words=list_stop_words)
tfidf = TfidfTransformer()

x_train_counts = count_vect.fit_transform(train_set)
x_train = tfidf.fit_transform(x_train_counts)

x_test_counts = count_vect.transform(test_set)
x_test = tfidf.transform(x_test_counts)
```

CountVectorizer[2] crea un oggetto che permette di trasformare testi in vettori di termini (bag of words). Applica diverse fasi della pipeline NLP (tokenizzazione, lowercase, eliminazione delle stop words…) e permette di creare un vocabolario utilizzabile per rappresentare tutti i testi nello stesso formato.

I parametri ngram_range, max_features e max_df vengono utilizzati per indicare gli n-grammi considerati nella tokenizzazione, la numerosità massima di features e un limite alto di frequenza dei termini (utile per eliminare le stop words).

Per creare il vocabolario basterà usare il metodo **fit_transform**. Dopo aver creato un vocabolario (o averlo fornito in input alla creazione dell'istanza) basterà usare il metodo **fit** per rappresentare altri testi usando quel particolare vocabolario.

TfidfTransformer[3] invece crea un oggetto che permette di applicare la normalizzazione tf-idf a vettori di bag of words.

Anche questa classe fa uso dei metodi **fit_transform** e **fit**. Le considerazioni sono identiche:

- **fit_transform** viene usato per creare un vocabolario usando i dati forniti in input
- **fit** viene usato quando abbiamo già fornito un vocabolario in precedenza

[2] Maggiori informazioni su: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[3] Maggiori informazioni su: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

Confondere la posizione dei comandi o utilizzarli in maniera impropria (per esempio usando solo il metodo **fit_transform**) porterà a funzionamenti errati del software (cancellazione del vocabolario e, in certi casi, crash del programma).

Dentro le variabili **x_train** e **x_test** avremo quindi dei vettori di termini normalizzati utilizzabili da un qualunque classificatore.

Classificatore: Regressione Logistica

Il primo classificatore che vedremo è il classificatore basato su **regressione logistica**. Per poterlo usare dovremo importare la classe **LogisticRegression**[4].

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(solver='liblinear')

log.fit(x_train, y_train_set)

y_train_preds = log.predict(x_train)
y_test_preds = log.predict(x_test)
```

LogisticRegression crea un oggetto utilizzabile nella classificazione di esempi.

La prima operazione da fare dopo aver creato il classificatore (log) e aver trasformato i dati attraverso un algoritmo di rappresentazione (Bag of Words o simili) è **fit**. **Fit** prende in input i dati (nel nostro caso i tweet) e le etichette attese (gli username) del training set e fissa i parametri del classificatore.

Successivamente si può effettuare la classificazione dei dati sia del training set che del test set usando il metodo **predict**.

Misura di Performance: Score F_1

Per valutare l'operato del classificatore abbiamo bisogno di una misura di performance. Useremo lo **score F_1** in quanto uno dei più precisi e slegato dalla conformazione dei set. Per calcolarlo useremo il metodo **f1_score** di **sklearn.metrics**.

```
from sklearn.metrics import f1_score
print("F1 training scores:
{:0.2f}".format(f1_score(y_train_set,y_train_preds,average='weighted')))
print("F1 test scores:
{:0.2f}".format(f1_score(y_test_set,y_test_preds,average='weighted')))

Out:
F1 training scores: 0.92
F1 test scores: 0.86
```

[4] Maggiori informazioni su: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

y=realDonaldTrump top features

Weight[?]	Feature
+4.014	russia
+3.812	democrats
+3.530	mueller
+3.529	collusion
+3.370	impeachment
+3.337	great
+3.093	witch
+3.014	witch hunt
+3.014	hunt
+2.954	hoax
+2.459	rt
+2.384	dems
+2.238	nothing
+2.188	fake
+2.090	russian
+2.066	big
+1.994	thank
+1.989	democrat
+1.786	never
+1.664	fbi
+1.663	pelosi
+1.662	report
... 6107 more positive ...	
... 3854 more negative ...	
-1.765	ll
-1.775	every
-1.801	kids
-1.803	join
-1.842	democracy
-1.863	let
-1.912	children
-1.930	families
-1.943	need
-1.982	trump
-2.000	re
-2.002	donald
-2.172	make
-2.431	today
-2.551	vote
-2.767	can
-3.221	women
-4.862	hillary

Come viene evidenziato dall'output di questa sezione di codice, il punteggio ottenuto dal classificatore in entrambi i set è considerevolmente alto. Possiamo dire che la capacità del modello è sufficiente (non c'è overfitting né underfitting).

Utilizzando particolari librerie come **eli5** si può inoltre anche capire come vengono pesati i vari token attraverso il metodo **show_weights**.

L'immagine ci mostra esattamente come ciò avviene: essendo un caso in cui abbiamo due etichette viene mostrata una singola colonna riguardante i token più usati da Trump (parte verde della colonna) e quelli più usati da Hillary Clinton (parte rossa). Per ogni feature viene indicato un valore numerico (chiamato peso) che viene usato dal classificatore per predire un'etichetta

Porteremo successivamente un esempio che aiuterà il lettore a capire in che modo il classificatore usa i pesi delle features per effettuare le predizioni.

Classificazione di Tweet

Proviamo a fornire al classificatore due tweet che non conosce:

```
source_test = [
    "The presidency doesn't change who you are. It reveals who you are. And we've seen all we need to of Donald Trump.",
    "Crooked Hillary is spending tremendous amounts of Wall Street money on false ads against me. She is a very dishonest person!"
]
```

Il primo tweet dovrebbe essere scritto da Hillary Clinton, il secondo da Trump.

Per poter classificare questi due tweet dobbiamo (dopo il pre-processing) trasformarli in vettori normalizzati come fatto precedentemente con il dataset. Perciò riutilizzeremo le stesse istanze già usate di **CountVectorizer** e **TfidfTransformer** (sincerandoci di usare così lo stesso dizionario). Successivamente useremo il classificatore per stabilire chi avrebbe potuto scrivere quei due tweet usando il metodo **predict_proba**.

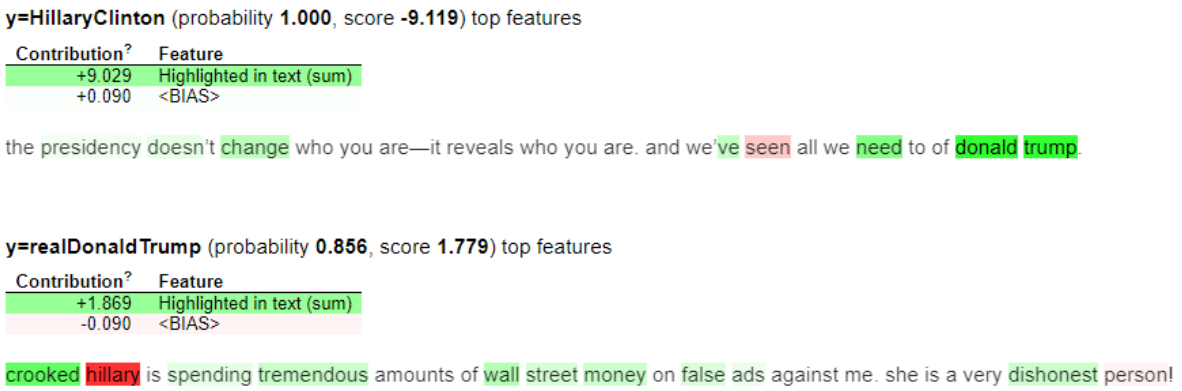
```
source_test = cleanTweet(source_test)
unknown_tweets_counts = count_vect.transform(source_test)
unknown_tweets = tfidf.transform(unknown_tweets_counts)
pd.DataFrame(log.predict_proba(unknown_tweets), columns=log.classes_)
```

	HillaryClinton	realDonaldTrump
0	0.9995	0.0005
1	0.2148	0.7852

Predict_proba ritorna le probabilità che l'esempio appartenga ad un'etichetta, per ogni etichetta presente nel nostro modello sottoforma di tabella. La somma di tutte le probabilità di ogni riga della tabella fa 1, confermando che siano probabilità lecite.

Nella fattispecie il classificatore associa il primo tweet ad Hillary con un ottimo 99% di probabilità e il secondo a Trump con un 78% di probabilità. Il modello sembrerebbe funzionare bene anche con dati sconosciuti.

Come per la classificazione, **eli5** provvede anche un metodo (**show_predictions**) per spiegare il perché si ottiene questo risultato. Nella fattispecie otteniamo i seguenti output:



Nel primo caso viene classificata la Clinton perché fa uso di parole che usa spesso come "donald", "trump" e "need".

Nel secondo si riconosce Trump. A differenza del primo, però, non abbiamo il 100% di probabilità per il semplice fatto che Trump usa una parola ("hillary") che il classificatore non associa a lui (ma alla Clinton), e quindi il classificatore penalizza lo score generale abbassando la probabilità che sia Trump.

La cosa interessante di questo tool (che purtroppo non è possibile apprezzare appieno in un file pdf) è che ci permette di avere una rappresentazione grafica dei termini che più hanno pesato nel calcolo del punteggio. Più è verde e più il termine indica che ci si riferisce all'etichetta mostrata (ovvero è una parola molto usata dalla persona indicata); viceversa più è rosso più si penalizza il punteggio (perché è un termine più comune per l'altra etichetta essendo questo un caso binario). Lasciando il puntatore del mouse su una delle parole colorate, si può anche vedere il peso della feature in questione.

Oltre a classificare tweet sconosciuti (inventati o prelevati attraverso mining da altri profili di Twitter o altre fonti), un'altra cosa molto interessante che questo modello permette di fare è quella di stampare tutte le probabilità predette dei tweet del dataset.

Tutte queste prove ci possono fare arrivare ad una conclusione: il classificatore viene allenato a riconoscere come scrivono gli utenti. Perciò potrebbe essere utilizzato nello scoprire **fake news** in relazione ai **fake account**, ovvero account fittizi che emulano account ufficiali per mandare messaggi fuorvianti (oppure riconoscere se un account legittimo e ufficiale sia stato "hackerato"). Ciò è possibile addestrando il nostro classificatore a riconoscere particolari utenti e sottomettergli i tweet che vogliamo classificare e che vogliamo riconoscere come "fake" o "legittimi". Ovviamente rientriamo nel campo dell'incertezza, ma un buon classificatore può essere utile in tal senso.

Classificazione con 3 (o più) etichette

Nel caso della regressione logistica, la presenza di più etichette comporta un complicarsi della logica che sta alla base. Fortunatamente, le librerie che utilizziamo implementano già tutto quello che ci serve per gestire il caso di più di due etichette nel dataset.

In particolare, **LogisticRegressor** implementa uno dei tanti algoritmi di classificazione logistica con più di due etichette: l'algoritmo **one-vs-rest**. (o **one-vs-rest**). Per spiegarlo in breve, questo algoritmo considera tanti regressori logistici quante sono le etichette e per ognuno effettua una classificazione binaria dove considera come etichette la particolare etichetta a cui è associato il regressore in questione e la somma degli insiemi di dati delle altre etichette. Usando questo metodo si può quindi mantenere la logica intrinseca della regressione logistica applicata alla classificazione. Per implementare questo algoritmo basterà impostare esplicitamente il parametro **multi_class** di **LogisticRegression** con il valore **'ovr'**. Anche in questo caso possiamo stampare la tabella delle features pesate per etichetta. Stavolta vedremo tre colonne in quanto abbiamo più di 2 etichette e non ci troviamo in un semplice caso binario.

y=BarackObama top features		y=HillaryClinton top features		y=realDonaldTrump top features	
Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature
+8.416	president obama	+7.087	hillary	+5.255	russia
+7.559	obama	+4.749	trump	+4.662	democrats
+4.386	president	+3.625	vote	+4.459	collusion
+3.062	read	+2.793	children	+4.309	great
+2.824	change	+2.728	women	+4.291	mueller
+2.576	leaders	+2.681	donald	+4.159	impeachment
+2.493	add	+2.547	can	+3.615	witch
+2.377	add name	*... 6271 more positive ...*		+3.564	hoax
+2.340	health	*... 13710 more negative ...*		+3.481	witch hunt
+2.240	check	-2.260	big	+3.481	hunt
... 5640 more positive ...		-2.316	impeachment	+3.321	thank
... 14141 more negative ...		-2.411	hoax	+3.255	rt
-2.226	vote	-2.440	witch hunt	+2.868	fake
-2.291	great	-2.440	hunt	+2.841	russian
-2.334	impeachment	-2.518	witch	+2.723	nothing
-2.500	election	-2.785	mueller	+2.711	dems
-2.619	democrats	-2.804	democrats	+2.585	amp
-2.823	thank	-3.053	russia	*... 10049 more positive ...*	
-2.986	russia	-3.193	collusion	*... 9932 more negative ...*	
-4.305	amp	-3.540	president	-3.320	hillary
-5.193	trump	-5.869	president obama	-3.441	obama
-5.205	hillary	-6.002	obama	-4.672	president obama

Effettuiamo nuovamente il test della predizione di tweet sconosciuti fornendo al classificatore i seguenti tweet:

```
source_test = [
    "The presidency doesn't change who you are. It reveals who you are. And we've seen all we need to of Donald Trump.",
    "Crooked Hillary is spending tremendous amounts of Wall Street money on false ads against me. She is a very dishonest person!",
    "Here are a couple of articles that are worth reading — from housing to health, these stories use data to explore challenges we face and demonstrate how policy solutions along with civic engagement can make a real difference in people's lives."
]
```

La tabella delle probabilità prodotta è:

	BarackObama	HillaryClinton	realDonaldTrump
0	0.00093	0.99889	0.00018
1	0.00004	0.40643	0.59352
2	0.44317	0.55675	0.00007

Rispetto a prima si sono abbassate le probabilità in merito ai primi due tweet: il classificatore indovina nuovamente chi sia il "legittimo" autore, ma con meno certezza. La causa di ciò è da collegare al fatto che abbiamo introdotto una nuova etichetta: questo ha infatti comportato l'aumento di incertezza.

Possiamo nuovamente valutare la scelta del classificatore stampando la tabella delle features e la rappresentazione del testo con i termini più usati:

y=BarackObama (probability **0.001**, score **-6.977**) top features

Contribution?	Feature
... 3 more positive ...	
... 4 more negative ...	
-0.926	<BIAS>
-5.471	Highlighted in text (sum)

the presidency doesn't change who you are—it reveals who you are. and we've seen all we need to of donald trump.

y=HillaryClinton (probability **0.999**, score **9.803**) top features

Contribution?	Feature
+9.314	Highlighted in text (sum)
... 4 more positive ...	
... 3 more negative ...	

the presidency doesn't change who you are—it reveals who you are. and we've seen all we need to of donald trump.

y=realDonaldTrump (probability **0.000**, score **-8.614**) top features

Contribution?	Feature
... 2 more positive ...	
... 5 more negative ...	
-0.892	<BIAS>
-6.791	Highlighted in text (sum)

the presidency doesn't change who you are—it reveals who you are. and we've seen all we need to of donald trump.

Per non perderci in dettagli mostreremo solo il primo esempio (ma il ragionamento vale per tutti e tre i casi): il tool di eli5 mostra il punteggio per ogni singola etichetta. Nel caso binario (due etichette) non è necessario perché ogni termine verde identifica maggiormente l'etichetta che viene predetta e i termini rossi si riferiscono all'etichetta meno probabile. Nel caso di tre o più etichette, invece, viene mostrato il calcolo del punteggio per ogni singola etichetta. Questo avviene perché i termini che identificano un'etichetta potrebbero (in misura minore o maggiore) identificare o meno anche un'altra. La predizione in questo caso viene fatta facendo la somma dei pesi di ogni feature riconosciuta nei testi, cosa che avviene anche nel caso binario, ma qua assume un'importanza maggiore, non banale. Infatti, può capitare di vedere due o più etichette la cui somma dei termini di peso positivo si assomiglia: in quel caso la discriminante sarà la somma dei termini di peso negativo. Essenzialmente potremmo trovare un caso come quello dell'immagine. Barack Obama fa uso di "change" (5° parola nella tabella delle features), ma la presenza del termine "trump" (penultima parola della tabella delle features) oltre che del termine "donald" fanno sì che il classificatore riconosca correttamente che Barack Obama non avrebbe potuto scrivere quel tweet.

Classificatore: Multinomial Naive Bayes

Un altro classificatore fornito è **MNB** (Naive Bayes multinomiale). Lo si ottiene importando **MultinomialNB** da **sklearn.naive_bayes**. Utilizza lo stesso formato di rappresentazione dei dati del regressore lineare.

Classificatore: KNN

Infine, il terzo classificatore fornito è **KNN** (K-nearest neighbors). Lo possiamo ottenere importando la classe **KNeighborsClassifier** da **sklearn.neighbors**. Valgono le stesse considerazioni fatte per il regressore logistico e MNB per quanto concerne la rappresentazione dei tweet. La differenza sostanziale Per questo classificatore è la presenza di un **iperparametro**: K. Gli iperparametri sono valori/impostazioni che influenzano le prestazioni del nostro classificatore e che vanno impostati oculatamente. Non vengono modificati durante il fitting del modello, ma vanno valutati in un'altra fase.

Per questo motivo dovremo effettuare un ulteriore split del dataset creando un terzo set di dati: il **validation set**. Il validation set viene usato per fissare il miglior valore di K che aumenta la misura di performance del classificatore. Nel codice questa fase di scelta del valore di K viene affrontata nella seguente maniera:

```
best_score = 0
best_k = 0

for k_value in range(1,11):
    knn = KNeighborsClassifier(n_neighbors=k_value)
    knn.fit(x_train, y_train_set)
    y_vali_preds = knn.predict(x_vali)
    print("{} - F1 Validation Score:
{:0.2f}".format(k_value,f1_score(y_vali_set,y_vali_preds,average='weighted'
)))
    if f1_score(y_vali_set,y_vali_preds,average='weighted') > best_score:
        best_score = f1_score(y_vali_set,y_vali_preds,average='weighted')
        best_k = k_value
```

Il software prova ad uno ad uno i valori di K (da 1 a 10) e sceglie quello con il miglior score F_1.

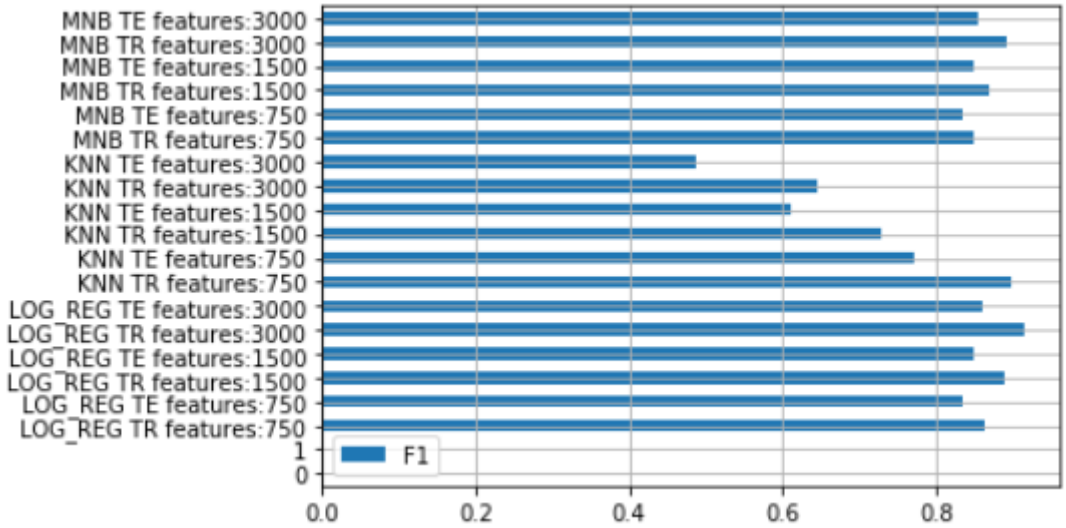
Una volta scelto il valore di K più adatto, lo si utilizza per istanziare il classificatore indicandolo come valore del parametro **n_neighbors** di **KNeighborsClassifier** e si può procedere con il fitting dei dati sul training set.

Sia KNN che MNB non hanno bisogno di particolari settaggi per operare in situazioni di dataset con più di due etichette.

Valutazione delle prestazioni

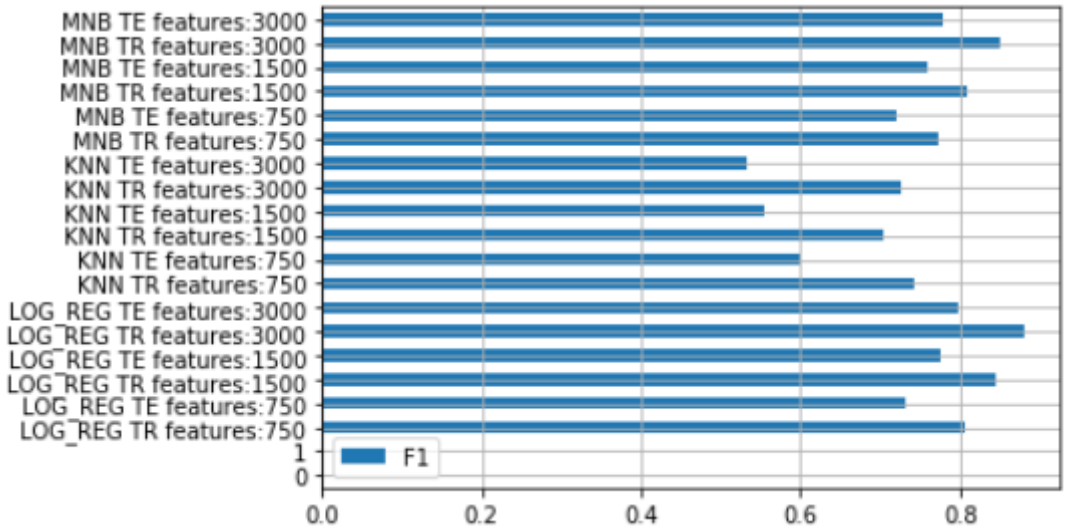
Utilizzando lo stesso dataset (e quindi uguali training e test set) e le stesse impostazioni in merito a numero di features e numero di etichette (fissando k a 3 per KNN), si è voluto fare un'indagine statistica per valutare le prestazioni dei tre classificatori. Il risultato dell'indagine viene riassunto con le seguenti immagini.

1) **Due Etichette [Clinton, Trump]**



Tra i tre il migliore in termini di prestazioni generali è il classificatore basato su regressione logistica, sebbene MBN si comporta molto bene. KNN è quello su cui si riscontra maggiore difficoltà, difficoltà che sembrano aumentare all'aumentare del numero di features.

2) **Tre Etichette [Clinton, Trump, Obama]**



Si può notare che le prestazioni generali di tutti gli algoritmi calano vistosamente, ma questo non ci deve far sorprendere: aggiungendo una terza etichetta abbiamo aggiunto maggiore incertezza, e quindi degradato le prestazioni dell'algoritmo.

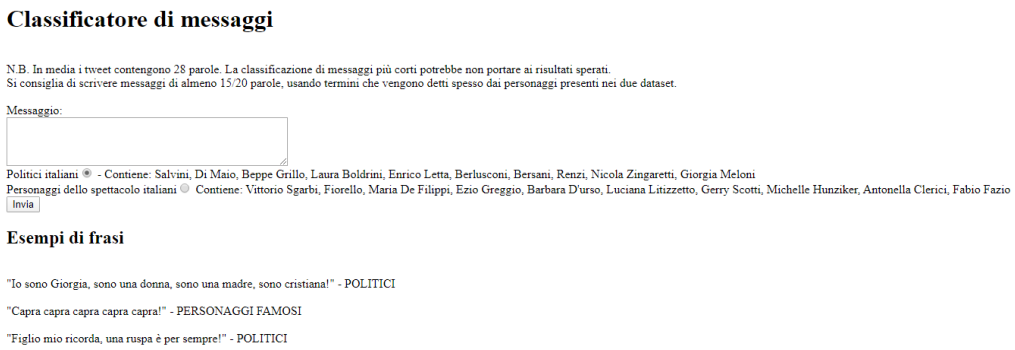
Interfaccia web

Un'applicazione di quanto detto può essere la creazione di applicazioni a scopo "ludico" che permettono agli utenti di inserire del testo su un classificatore già allenato e ottenere l'output dell'etichetta più probabile associata a quel testo.

Nella fattispecie, è stata prodotta un'applicazione web che fornisce due classificatori allenati su due categorie di personaggi italiani: un classificatore sui politici e uno su personaggi dello spettacolo.

Anticipiamo che è stato utilizzato un classificatore basato su Regressione Logistica (scelta dettata dal fatto che è l'unico dei tre compatibile con eli5). Inoltre, il classificatore è già allenato: non viene quindi ricreato ogni volta che si interagisce col programma, ma viene semplicemente caricato dalla memoria per essere utilizzato direttamente sui dati che riceve in input.

Presentiamo adesso la schermata principale dell'applicazione.



E' possibile indicare un messaggio da classificare e una delle due categorie disponibili di etichette, formate da 10 personaggi entrambe i quali nomi sono mostrati nella pagina stessa.

Essendo che in media i messaggi di tweet contengono 28 parole, è consigliato l'inserimento di messaggi di una lunghezza di almeno 15/20 parole per aumentare la precisione delle predizioni.

Inserita la frase e scelto uno dei due dataset disponibili, si clicca sul tasto "Invia" e si attende l'esecuzione del server.

Per esempio, inseriamo la frase "Io sono Giorgia, sono una donna, sono una madre, sono cristiana!" e scegliamo il dataset dei "Politici italiani". L'output generato sarà:

Viene fornita un'immagine del personaggio che è stato predetto e viene indicato con che probabilità è stato riconosciuto il personaggio. Semplicemente viene indicato come personaggio predetto il più probabile dei 10 presenti nel dataset secondo le probabilità stabilite dal classificatore.

Oltre a questo output, vengono stampate anche:

- La tabella dei termini più usati e meno usati dalle etichette con i relativi pesi
- Per ogni etichetta, la tabella dei termini rilevati nella frase, i loro pesi e la somma dei pesi

Questi dati permettono di avere informazioni precise sul ragionamento alla base del classificatore e capire il motivo per cui è stato generato un particolare output.

Lista dei metodi del sorgente

In questo paragrafo ci concentreremo nella descrizione in dettaglio dei vari metodi e della classe TweetMiner usati nel sorgente dell'algoritmo del progetto. Di ognuno di essi spiegheremo l'uso che se ne fa, i parametri di input e i parametri di output eventuali.

Iniziamo con un elenco dei metodi:

Classe TweetMiner

mine_user_tweets(self, user, mine_retweets, no_replies, max_pages)

Sezione Generale

main()

classify(clas, tfidf, vect, dataset, api)

manualTweets()

apiTweets(api)

datasetTweets(dataset)

cleanTweet(raw_data)

getAPI(ck, cs, atk, ats)

setAPI()

makeData()

mine_tweets(username,api,number_tweets)

printDF()

getDF()

addUser(username,dataset)

dropUser(username,dataset,users_in_dt)

printDataset(dataset, low_lim, high_lim, user)

printPlot(dataset,s)

clean(dataset)

splitDataset(dataset,percent,knn,knn_percent)

LRegr(dataset, num_label, features, tdf)

KNN(dataset, features, tdf, k)

MNB(dataset, features, tdf)

chooseClassifier(dataset,num_label)

Lista delle dipendenze

Per poter far girare il codice sul proprio sistema bisogna aver già installato i principali pacchetti di Python (pandas, numpy, iPython display…). Inoltre bisognerà installare:

- Textacy
- Eli5
- Python-twitter

Per poter visualizzare correttamente i dati stampati da Eli5 bisognerà usare Jupyter Notebook in quanto i dati stampati sono formattati in HTML e non sono visualizzabili diversamente.

Descrizione dei metodi del sorgente

TweetMiner(self, api, result_limit = 100, max_tweets=3000)

Costruttore della classe TweetMiner.

Parametri:

- **api**, api autorizzate di Twitter (senza le quali non si potrebbe fare il mining)
- **result_limit,** numero di Tweet che vengono prelevati ogni volta che il metodo **mine_user_tweets** usa il comando **GetUserTimeline()** delle api di Twitter. Valore di default 100, valore massimo 200
- **max_tweets,** numero Massimo di Tweet che il TweetMiner scaricherà. Valore di default 3000

mine_user_tweets(self, user="", mine_retweets=False, no_replies=True, max_pages=40)

Metodo interno della classe TweetMiner che si occupa di fare il mining di tweet di un particolare utente. Torna una lista di dizionari di tweet.

Parametri:

- **user,** utente da cui prelevare i tweet. Valore di default la stringa vuota “”
- **mine_retweets,** valore booleano che indica se si vuole prelevare anche i retweet dell’utente. Valore di default False
- **no_replies,** valore booleano che indica se si vuole escludere le risposte ai tweet dell’utente. Valore di default True
- **max_pages,** numero massimo di richieste consecutive che verranno effettuate. Valore di default 40

main()

Metodo iniziale che avvia il software. Permette di interagire con una GUI testuale che permette di fare diverse operazioni specifiche come:

- Inizializzare le API
- Mining dei tweet di utenti di Twitter
- Aggiungere o eliminare utenti dal dataset
- Scegliere il classificatore e tweet interni o esterni al dataset

classify(clas, tfidf, vect, dataset, api)

Utilizzato per effettuare la classificazione di tweet interni o esterni al dataset. Non torna alcun valore, ma stampa a schermo il risultato della classificazione effettuata.

Parametri:

- **clas,** classificatore tra i tre presentati (Log_Reg, KNN, MBN)
- **tfidf,** istanza di TfidfTransformer
- **vect,** istanza di CountVectorizer
- **dataset,** il DataFrame generale non splittato
- **api,** istanza autentificata delle api Twitter

manualTweets()

Usato dentro il metodo **classify**. Permette di inserire manualmente un numero variabile di tweet e ritorna un DataFrame.

apiTweets(api)

Usato dentro il metodo **classify**. Permette di inserire un numero variabile di tweet prelevandoli da Twitter (chiede di fornire le username) e ritorna un DataFrame.

Parametri:

- **api,** istanza autentificata delle api Twitter

datasetTweets(dataset)

Usato dentro il metodo **classify**. Permette di inserire un numero variabile di tweet prelevandoli dal dataset (chiede di fornire il numero della riga) e ritorna un DataFrame.

Parametri:

- **dataset,** il DataFrame generale non splittato

cleanTweet(raw_data)

Metodo usato per effettuare preprocessing dei tweet ed eliminare elementi di testo non utilizzati nella rappresentazione dei tweet. Torna una lista di stringhe.

Parametri

- **raw_data,** lista di stringhe

getAPI(ck, cs, atk, ats)

Metodo usato per inizializzare le API di Twitter. Torna l'istanza di api o None.

Parametri:

- **ck,** stringa
- **cs,** stringa
- **atk,** stringa
- **ats,** stringa

setAPI()

Metodo usato insieme a getAPI() per inizializzare le api di Twitter. Chiede di inserire i quattro token richiesti per autentificare l'accesso alle api. Torna l'istanza di api o None.

makeData()

Metodo usato per creare la cartella "DATA", usata per salvare i tweet che si prelevano da Twitter.

mine_tweets(username,api,number_tweets=100)

Metodo usato per gestire il mining dei tweet di un utente di Twitter. Controlla se non esiste in memoria un file contenente i dati richiesti e chiede se sovrascriverlo. Eventualmente istanzia un oggetto di tipo TweetMiner e lo usa per creare una nuova banca dati di tweet dell'utente indicato salvando i dati in un file csv dentro la cartella "DATA".

Parametri:

- **username,** stringa dell'utente da cui prelevare i tweet
- **api,** istanza autentificata delle api Twitter
- **number_tweets,** intero usato nell'istanziazione di TweetMiner. Default 100

printDF()

Metodo usato per stampare la lista degli utenti per cui esiste un file csv in memoria

getDF()

Metodo usato per ritornare la lista degli utenti per cui esiste un file csv in memoria

addUser(username,dataset)

Metodo usato per aggiungere tutti i tweet di un utente al dataset dei dati prelevando i dati dalla memoria o da Twitter. Torna il nuovo dataset e un valore intero che indica eventuali errori di esecuzione.

Parametri:

- **username,** stringa dell'utente da cui prelevare i tweet
- **dataset,** il DataFrame generale non splittato

dropUser(username,dataset,users_in_dt)

Metodo usato per eliminare tutti i tweet di un utente dal dataset. Torna il nuovo dataset.

Parametri:

- **username,** stringa dell'utente da cui prelevare i tweet
- **dataset,** il DataFrame generale non splittato
- **users_in_dt,** lista di stringhe. Contiene la lista degli utenti i quali tweet sono presenti sul dataset

printDataset(dataset, low_lim = 0, high_lim = 0, user = None)

Stampa il dataset dei tweet non splittato.

Parametri:

- **dataset,** il DataFrame generale non splittato

- **low_lim,** intero che indica l'indice basso dal quale stampare. Default 0
- **high_lim,** intero che indica l'indice alto dal quale stampare. Default 0
- **user,** stringa che indica l'utente del quale vogliamo stampare i tweet che abbiamo sul dataset. Default None

printPlot(dataset,s='')

Metodo usato per stampare il plot dei dati del dataset e dei set splittati (training, test e validation set).

Parametri:

- **dataset,** il DataFrame dei dati (splittato e non)
- **s,** stringa usata per stampare a video il nome del set che si sta per plottare. Default ""

clean(dataset)

Metodo che applica operazioni di preprocessing sui tweet del dataset. Ritorna un DataFrame.

Parametri:

- **dataset,** il DataFrame splittato (training, validation, test set) o fornito dall'utente

splitDataset(dataset,percent=0.25,knn=False,knn_percent=0.33)

Metodo che divide un DataFrame in più parti randomicamente. Torna dalle 4 alle 6 liste in base al valore di knn. Usato prima del fitting del modello.

Parametri:

- **dataset,** il DataFrame generale non splittato
- **percent,** double che indica la percentuale di dati del dataset che popoleranno il test set. Default 0.25
- **knn**, booleano che indica se il classificatore usato è KNN. Se è True viene generato il Validation Set e l'output del metodo passa da 4 elementi a 6 elementi. Default False
- **knn_percent**, double che indica la percentuale di dati del training set che popoleranno il validation set. Default 0.33

LRegr(dataset, num_label, features=3000, tdf=True)

Metodo che permette di allenare un classificatore basato su regressione logistica. Torna il classificatore, l'istanza di TfidfTransformer e l'istanza di CountVectorizer.

Parametri:

- **dataset,** il DataFrame generale non splittato
- **num_label,** intero che indica il numero di etichette presenti nel dataset. Se maggiore di 2 il classificatore usa
  all-vs-rest
- **features,** intero che indica il numero di feature massimo per ogni rappresentazione. Default 3000
- **tdf,** booleano che indica se applicare la normalizzazione tf-idf. Default True

KNN(dataset, features=750, tdf=False, k=0)

Metodo che permette di allenare un classificatore basato su K-Nearest Neighbours. Torna il classificatore, l'istanza di TfidfTransformer e l'istanza di CountVectorizer.

Parametri:

- **dataset,** il DataFrame generale non splittato
- **features,** intero che indica il numero di feature massimo per ogni rappresentazione. Default 3000
- **tdf,** booleano che indica se applicare la normalizzazione tf-idf. Default True
- **k**, intero che indica il valore dell'iperparametro k. Default 0. Se è 0 viene scelto un valore compreso in [1,10] che massimizza le prestazioni del classificatore nel validation set

MNB(dataset, features=3000, tdf=True)

Metodo che permette di allenare un classificatore basato su Multinomial Naive Bayes. Torna il classificatore, l'istanza di TfidfTransformer e l'istanza di CountVectorizer.

Parametri:

- **dataset,** il DataFrame generale non splittato
- **features, intero che** indica il numero di feature massimo per ogni rappresentazione. Default 3000
- **tdf,** booleano che indica se applicare la normalizzazione tf-idf. Default True

chooseClassifier(dataset,num_label)

Metodo che chiede all'utente quale dei tre classificatori utilizzare. In base alla scelta dell'utente viene ritornato il classificatore richiesto, l'istanza di TfidfTransformer e l'istanza di CountVectorizer.

Parametri:

- **dataset,** il DataFrame generale non splittato
- **num_label,** intero che indica il numero di etichette presenti nel dataset