SD-Pràctica 2

Alex Soriano Faiges

Arnau Francesc Llaberia Declara

Índex

1. Recollida de informació

Per a recollir la informació de internet, hem utilitzat dos vies, la primera, per Twitter, amb la utilització de la llibreria Tweepy, i la segona via Reddit, amb la pertinent utilització de la llibreria Web Crawler.

1.1. Tweepy

Primer de tot, per a utilitzar tweepy, que no deixa de ser la API que ens dona twitter per a la recollida d'informació d'aquest, hem de donar d'alta com developers el nostre compte; així aconseguirem les nostres claus d'accés que després utilitza per a fer servir la llibreria.

```
def get_auth():
    auth = tweepy.OAuthHandler("[redacted]", "[redacted]")
    auth.set_access_token("[redacted]", "[redacted]")
    return auth
```

Després d'autenticar-nos, utilitzem la funció *tweepy.Cursor* per buscar els tweets amb la característica que busquen els tweets més recents segons la paraula passada per paràmetre i el idioma que volem (També inicialitzem el *SentimentIntensityAnalyzer*).

```
auth = get_auth()
api = tweepy.API(auth, wait_on_rate_limit=True)

analyzer = SentimentIntensityAnalyzer()
qstring=word+" lang=ca OR lang:es"
```

Per cada tweet, guardem en un diccionari el text, la url, el sentiment del tweet (mitjançant la llibreria vaderSentiment), la data, el lloc d'on es fa, el dispositiu que s'ha utilitzat i el en quina llengua està el tweet; ho comprimim en format json, ho pengem al bucket de IBM.

```
for status in tweepy.Cursor(api.search, q=qstring ,tweet_mode="extended").items(500): #numberOftwets

    #print("----------------------------------------------------------------------------------------------------------------------------------------------------------")
    textos.append(status.full_text)
    urls.append("https://twitter.com/twitter/statuses/"+str(status.id)+",")
    sentiments.append(str(analyzer.polarity_scores(textos[i])['compound']))
    dates.append(status.created_at.strftime("%m/%d/%Y %H:%M:%S"))
    local.append(str(status.user.location))
    source.append(str(status.source))
    lenguaje.append(str(status.lang))
    i += 1;

datos = {
    "Mensaje": textos,
    "url": urls,
    "sentiment": sentiments,
    "date": dates,
    "local": local,
    "source": source,
    "lenguaje": lenguaje
}

now = datetime.now()
data = now.strftime("%m/%d/%Y")

storage = Storage()
storage.put_object(BUCKET,data+"-"+word+".json",json.dumps(datos))
```

1.2. Web Crawler

Per el web crawler, haurem de crear una classe amb el paràmetre *scrapy.Spider*. Aquesta funció de la llibreria ens servirà per recórrer el codi html de la pàgina.

```
class Reddit (scrapy.Spider):
```

Al executar la classe, començarem amb la funció *start_requests*; aquesta funció començarà la recollida de codi de la pàgina web que li passem.

```
def start_requests(self):
    #generate API URL
    url = self.base_url + urlencode(self.params)

    #make initial HTTP request
    yield scrapy.Request(
        url=url,
        callback=self.parse_page
    )
```

Això cridarà a la funció *parse_page*, que serà la que realment farà la recollida del codi.

En aquesta funció el que farem serà passar directament el html a json, i per cada json que trobem, agafarem els links dels posts que trobarem a Reddit, per a poder centrar-nos en la noticia en qüestió, fent que aquesta cridi a la funció *parse_post*, on tractarem la funció.

Abans de passar al anàlisi de la funció anomenada, parlarem sobre el que fem per a que la crida de scrapy sigui sol del html original, ja que si no agafaríem sol 8 o 9 notícies; el que fem és actualitzar la url a buscar, per a que ens doni el codi següent, com si de forma natural amb el ratolí baixéssim la rodeta com sempre. Això fa que ens donin més notícies i crear una cerca de informació recursiva.

De forma final, tindríem un if_else statement, per a controlar les crides, on al acabar aquest if/else, agafi tots els valors que tenim emmagatzemats a un diccionari que actualitzem a la funció *parse_post* i el pengem al bucket.

```
def parse_post(self,response):
    #extract data
    posts = {
        'title': response.css('h1[class="_eYtD2XCVieq6emjKBH3m"]::text').get(),
        'likes': response.css('div[class="_1rZYMD_4xY3gRcSS3p8ODO _3a2ZHWaih05DgAOtvu6cIo"]::text').get(),
    }
    self.titol.append(response.css('h1[class="_eYtD2XCVieq6emjKBH3m"]::text').get())
    self.likes.append(response.css('div[class="_1rZYMD_4xY3gRcSS3p8ODO _3a2ZHWaih05DgAOtvu6cIo"]::text').get())
```

Aquí tindríem el codi on actualitzem el diccionari amb els valors de 'títol' i 'likes' de cada una de les notícies i seguidament el codi recursiu de *parse_page*.

```python
def parse_page(self,response):
    global BUCKET
    if (self.count<self.max) :
        json_data= json.loads(response.text)

        #loop over posts
        for post in json_data['posts']:
            posts_url = json_data['posts'][post]['permalink']

            # make HTTP request to the given post
            yield response.follow(
                url=posts_url,
                callback=self.parse_post
            )
            break
        #extract post urls
        # print(json_data['posts'])


        #update string query parameters
        self.params['after']= json_data['token']
        self.params['dist']= json_data['dist']

        #generate API URL
        url = self.base_url + urlencode(self.params)

        #update count
        self.count = self.count + 1

        #make recursive HTTP request to the next page
        yield scrapy.Request(
            url=url,
            callback=self.parse_page
        )
    else :
        storage = Storage()
        now = datetime.now()
        data = now.strftime("%m/%d/%Y")
        storage.put_object(BUCKET,data+"-"+"web.json",json.dumps(self.post))
```

Cal destacar que la llibreria scrapy sol ens deix un número finit de crides recursives i que si ens passem, para el codi directament, aleshores el que fem es posar un nombre que ens asseguri que funcionarà correctament, en el nostre cas, després de diverses proves, és 35.

```python
    max=35 #maximum value due to limitations of scrappy
```

També ensenyem els paràmetres de la crida http:

```python
base_url = 'https://gateway.reddit.com/desktopapi/v1/subreddits/COVID19?'

params = {
    "rtj":"only",
    "redditWebClient":"web2x",
    "app":"web2x-client-production",
    "allow_over18":"",
    "include":"prefsSubreddit",
    "after":"t3_nejiy5",
    "dist":"8",
    "layout":"card",
    "sort":"hot",
    "geo_filter":"ES"
}
```

2. Tractament de dades

Per tractar les dades, també diferenciarem per cada una de les llibreries utilitzades

2.1. Tweepy

Per a twitter, agafem el json del bucket, el passem de json a diccionari i de diccionari a *DataFrame*, on apartir d'aquí podrem treballar.

Primer actualitzem la columna de sentiment, on al estar en string, no li podrem treure un valor, però si la passem a float si.

Calculem dos gràfics: el primer comparem la quantitat de sentiments positius en contra dels negatius per mitja i el segon es la concentració del sentiment en cada tweet. Per a fer els gràfics utilitzem la llibreria *matplotlib*, on fer un subplot per ajuntar els dos gràfics.

```
def grafic_twitter(word):
    global BUCKET

    now = datetime.now()
    data = now.strftime("%m/%d/%Y")
    key= data+"-"+ word +'.json'

    storage = Storage()
    json_read = storage.get_object(BUCKET,key)
    data_analize = json.loads(json_read)
    df = pd.DataFrame.from_dict(data_analize,orient='columns')
    df["sentiment"]=df["sentiment"].astype(float)

    sent_pos=len(df[df["sentiment"] >= 0])
    sent_neg=len(df[df["sentiment"] < 0])
    sents=[sent_pos,sent_neg]
    noms=['Positiu','Negatiu']


    llista=list(df["sentiment"].astype(float))
    ocurrences = collections.Counter(llista)
    dictionary = ocurrences.items()
    dfa= pd.DataFrame.from_dict(dictionary,orient='columns')
    dfa.columns = ['value','quantity']
    dfa['quantity']= dfa['quantity'].astype(float)
    x_value=list(dfa["value"])
    y_value=list(dfa["quantity"])

    fig, (axs1, axs2)= plt.subplots(1,2)

    axs1.bar(noms,sents)
    axs1.set_title('Diferencia del sentiment')
    axs2.bar(x_value,y_value)
    axs2.set_title('Concentració del sentiment')
    fig.suptitle('Estudi de la paraula:'+word)
    # plt.bar(x_value,y_value)
```

2.2. Web Crawler

En aquest cas, al extreure menys informació, no tractem el sentiment. Primer fem el mateix que l'anterior cas per a tindre la informació correcte en un *DataFrame.* En aquest cas, tractem la mitja de interaccions (likes) per cada noticia, segons del que tracti, i la mostrem en una gràfica al igual que l'anterior apartat. Cal destacar que si per algun casual no s'hi parles d'alguna de les paraules clau que hem triat, hem de passar un filtratge per a així, en compte de que la mitja en surti el string 'nan', ens surti el 0, fent servir una petita funció anomenada *eliminateNaNs*.

```
def grafic_web():
    global BUCKET

    def eliminateNaNs(list) :
        i = 0
        for x in list :
            if str(x) == 'nan':
                list[i] = 0
            i+=1
        return list

    #Treatment of json to pandas
    now = datetime.now()
    data = now.strftime("%m/%d/%Y")
    key= data+'-web.json'

    storage = Storage()
    json_read = storage.get_object(BUCKET,key)
    data_analize = json.loads(json_read)
    df = pd.DataFrame.from_dict(data_analize,orient='columns')
    df["likes"]=df["likes"].astype(int)


    # average likes about key words
    likesSARS=df[df["titol"].str.contains("SARS-CoV")].mean()["likes"]
    likesModerna=df[df["titol"].str.contains("Moderna")].mean()["likes"]
    likesPfizer=df[df["titol"].str.contains("Pfizer")].mean()["likes"]
    likesAstra=df[df["titol"].str.contains("AstraZeneca")].mean()["likes"]
    likesSputnik=df[df["titol"].str.contains("Sputnik")].mean()["likes"]
    likesJan=df[df["titol"].str.contains("Janssen")].mean()["likes"]

    name = ['SARS','Moderna','Pfizer','AstraZeneca', 'Sputnik' , 'Janssen']
    allLikes = [likesSARS,likesModerna,likesPfizer,likesAstra, likesSputnik, likesJan]
    allLikes = eliminateNaNs(allLikes)

    plt.bar(name,allLikes)
    plt.show()
```

3. Execució al núvol

3.1. Tweepy

Per a executar al núvol, agafem cada un de les funcions principals i amb el *pool.map* executem al núvol. Cal destacar que en aquest cas hem hagut de separar en diferents crides, ja que si no, sobrepassa el temps màxim que ens permet la funció al núvol, què són 10 minuts.

```
if __name__ == '__main__':

    with Pool() as pool:
        pool.map(tweepy_scan,  [ "covid", "moderna"])
        pool.map(tweepy_scan,  [ "pfizer", "astrazeneca"])
        pool.map(tweepy_scan,  [ "sputnik v", "janssen"])

    grafic_twitter("covid")
    grafic_twitter("moderna")
    grafic_twitter("pfizer")
    grafic_twitter("astrazeneca")
    grafic_twitter("sputnik v")
    grafic_twitter("janssen")
```

3.2. Web crawler

En aquest cas, creem una petita funció per inicialitzar el *web crawler* i poder executar-ho sense problemes.

```
def function(data):
    process = CrawlerProcess()
    process.crawl(Reddit)
    process.start()

if __name__ == '__main__':

    with FunctionExecutor() as fexec:
        fut = fexec.call_async(function,"")

    grafic_web()
```