Facebook Scraping: scaricare tutti i post delle pagine Facebook

Da quando sono esplosi gli scandali relativi all'uso dei dati dei social network, come quello di Cambridge Analytica, Facebook ha messo in piedi un sistema di controllo delle applicazioni. In poche parole, ora qualsiasi applicazione voglia accedere a ogni tipo di dato degli utenti deve prima superare un controllo in cui, in teoria, Facebook dovrebbe verificare che l'app non usi i dati in modo contrario alle norme di condotta previste dal social network.
Il problema è che, al momento, il sistema non funziona bene: ovviamente è appena partito, e si presume che in futuro verrà migliorato, ma ha una serie di difetti fondamentali che saranno difficili da correggere a meno che Facebook non sia pronto a spendere davvero molte risorse finanziarie. Già adesso, infatti, ci sono delle persone incaricate di analizzare ogni app che viene sottoposta alla verifica, ma hanno migliaia di app da seguire e non hanno quindi il tempo di entrare nei dettagli. Soprattutto, nessuno controlla il codice sorgente delle app, quindi non c'è davvero una garanzia che questo controllo serva a impedire utilizzi impropri dei dati del social network. Allo stesso tempo, inoltre, i meccanismi di autorizzazione delle app offerti al momento sono insufficienti a coprire alcune delle app più legittime: parliamo di quelle che collezionano dati pubblici per realizzare statistiche (per pubblica amministrazione, università, e ricerca). Al momento è molto difficile farsi approvare una app di tipo desktop, l'opzione non è prevista e gli script non sono visti di buon occhio. Tuttavia, una università, un istituto di statistica, o una redazione giornalistica hanno in genere bisogno di accedere soltanto a dati come i vari post delle pagine dei personaggi famosi (per

analizzare il loro linguaggio e capire come cambi la comunicazione in caso di eventi importanti, o controllare la veridicità delle affermazioni). Un esempio semplice è un team di ricercatori che voglia controllare sistematicamente la percentuale di verità dei post di un politico, il cosiddetto fact checking. In questi casi si vuole accedere soltanto a dati che sono già pubblici, e che quindi possono essere raccolti senza violare la privacy di nessuno.

Uno script con Python

Per il nostro script utilizziamo Python3: nonostante sul sito ufficiale venga ancora presentato il vecchio Python2 per questioni di retrocompatibilità, la versione 3 presenta delle differenze importanti che rendono alcune funzioni incompatibili. Per essere sicuri di poter utilizzare lo script che presentiamo (alla fine dell'articolo trovare un link all'intero codice sorgente), bisogna installare sul proprio pc almeno la versione 3.6 di Python.

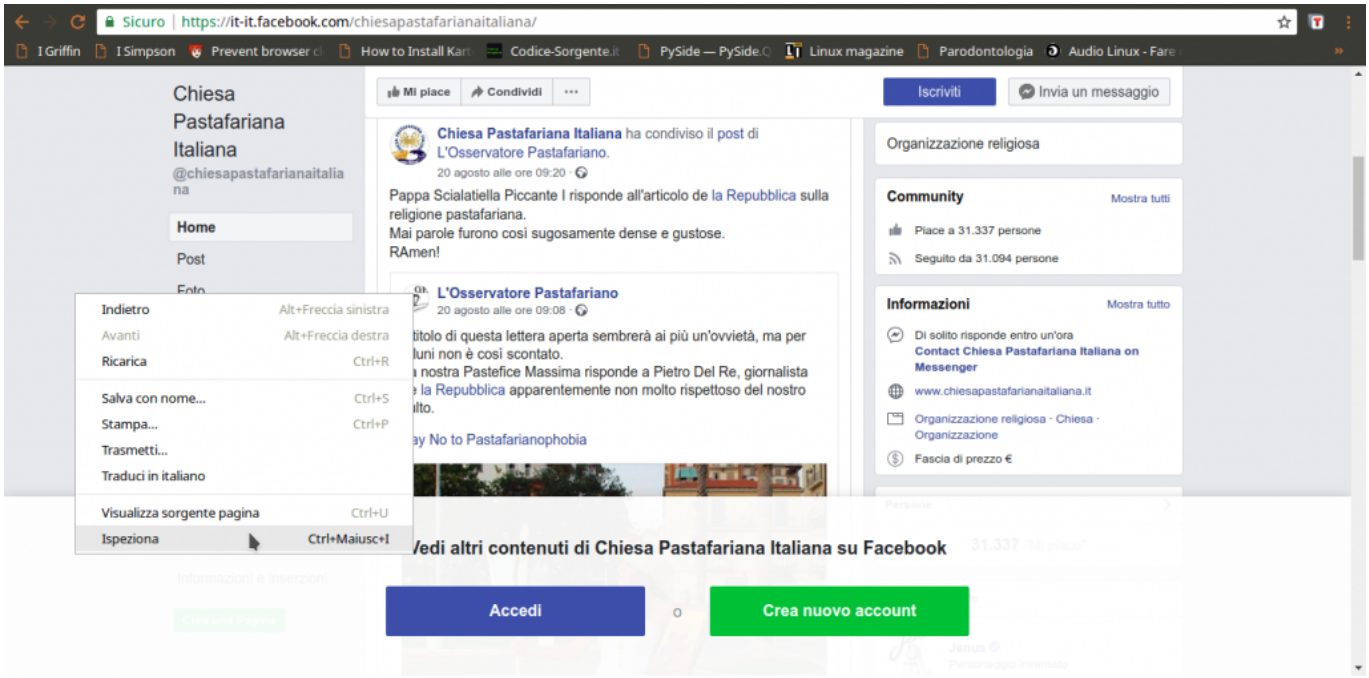
Abbiamo quindi pensato di realizzare uno script in Python che si occupi di eseguire lo scraping delle pagine Facebook pubbliche. Lo scraping è, per chi non lo sapesse, un insieme di tecniche di estrazione di informazioni da pagine web e altri documenti, in modo automatico, ripulendole da ciò che non serve. Un ricercatore universitario potrebbe scaricarsi i post di una pagina Facebook aprendola col browser e scorrendola verso il basso fino a visualizzarli tutti, selezionando il testo di ciascuno e copiandoselo. Ma sarebbe una operazione lunghissima e noiosa. Analizzando il codice delle pagine HTML che Facebook fornisce, invece, possiamo automatizzare l'estrazione dei testi (o delle immagini, se volete scaricarvi i meme delle vostre pagine preferite, basta modificare lo script per cercare i tag **img** invece dei tag **p**).

E ovviamente lo script che realizziamo non richiede alcun accesso alle API o approvazione da parte di Facebook, perché di fatto faremo la stessa cosa che fa ogni utente che vuole guardare una pagina Facebook, permettendoci di bypassare tutte le verifiche che Facebook ha messo in piedi per le app.
Non dobbiamo, infatti, dimenticare un concetto fondamentale: se una informazione è disponibile, c'è sempre un modo non ufficiale per ottenerla. Quando carichiamo una pagina Facebook in Google Chrome, l'interfaccia realizzata con HTML e Javascript carica solo un certo numero di post. Quando "scrolliamo" la pagina, scendendo verso il basso, deve esserci una qualche funzione che si accorge che stiamo scendendo e quindi richiede al server un certo numero di nuovi post da visualizzare. È abbastanza ovvio che questa richiesta debba essere fatta, dalla pagina HTML+JS, con una richiesta HTTP (usando il meccanismo Ajax, quindi la funzione xmlhttprequest di Javascript). Se ne deduce che da qualche parte all'interno della pagina ci deve essere un riferimento a un'altra pagina che fornisce un elenco di post da visualizzare. L'accesso a questi dati da parte di script automatici invece che dai normali browser web è una cosa che, a prescindere dai suoi sforzi, Facebook non potrà mai impedire.

Table of Contents

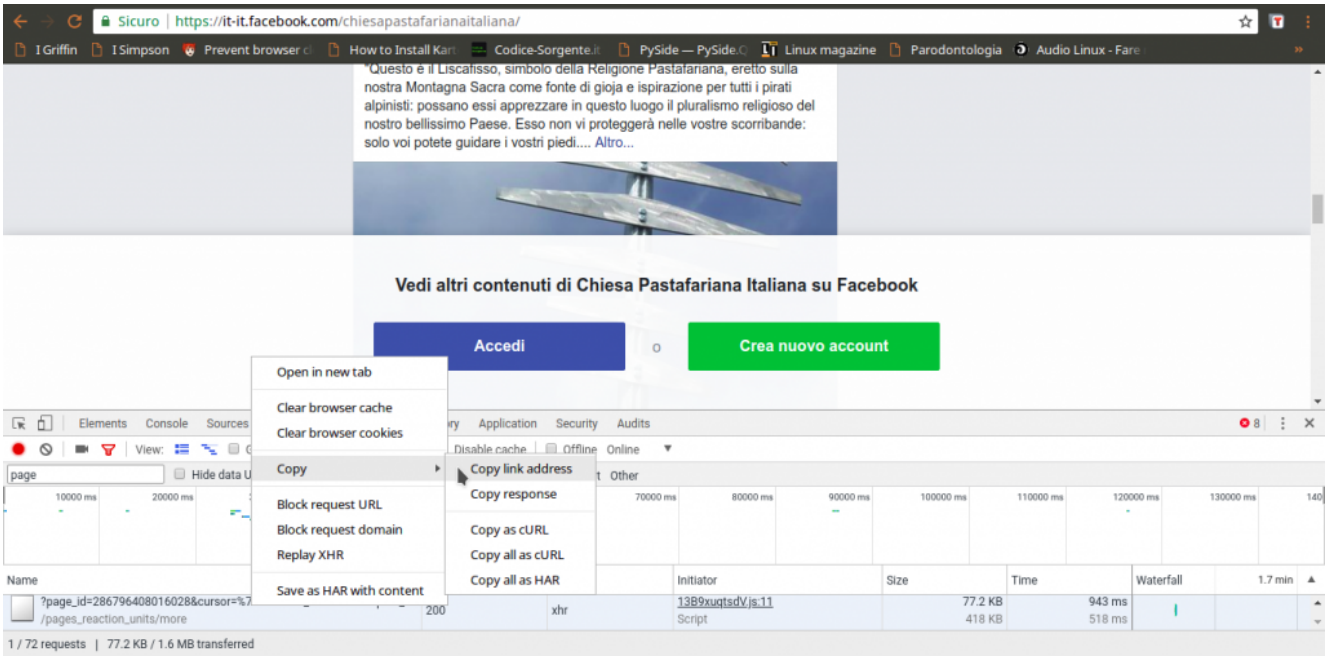
Scoprire l'indirizzo per ottenere i post

Per prima cosa dobbiamo scoprire come funzioni Facebook, cioè come vengano recuperati i vari post. In poche parole, bisogna conoscere il proprio obiettivo. Siccome si tratta di un sito web, la cosa migliore da fare è aprire una pagina Facebook (per esempio https://it-it.facebook.com/chiesapastafarianaitaliana/) col proprio browser, come Google Chrome, cliccando poi col tasto destro sulla pagina per scegliere la voce **Ispeziona**.



Tra le varie schede disponibili, quella che serve per capire cosa succeda è quella chiamata **Network**: si occupa di presentare in tempo reale le varie richieste HTTP che vengono fatte. Siccome è ovvio che la pagina di Facebook, per caricare altri post, abbia bisogno di fare una richiesta al server di Facebook per ottenerli, è anche ovvio che apparirà qui. Tutto quello che dobbiamo fare a questo punto è scorrere la pagina verso il basso, per obbligarla a caricare altri post: nel pannello vedremo comparire una richiesta a una pagina chiamata **page_reaction_units**. Sembra proprio che abbiamo

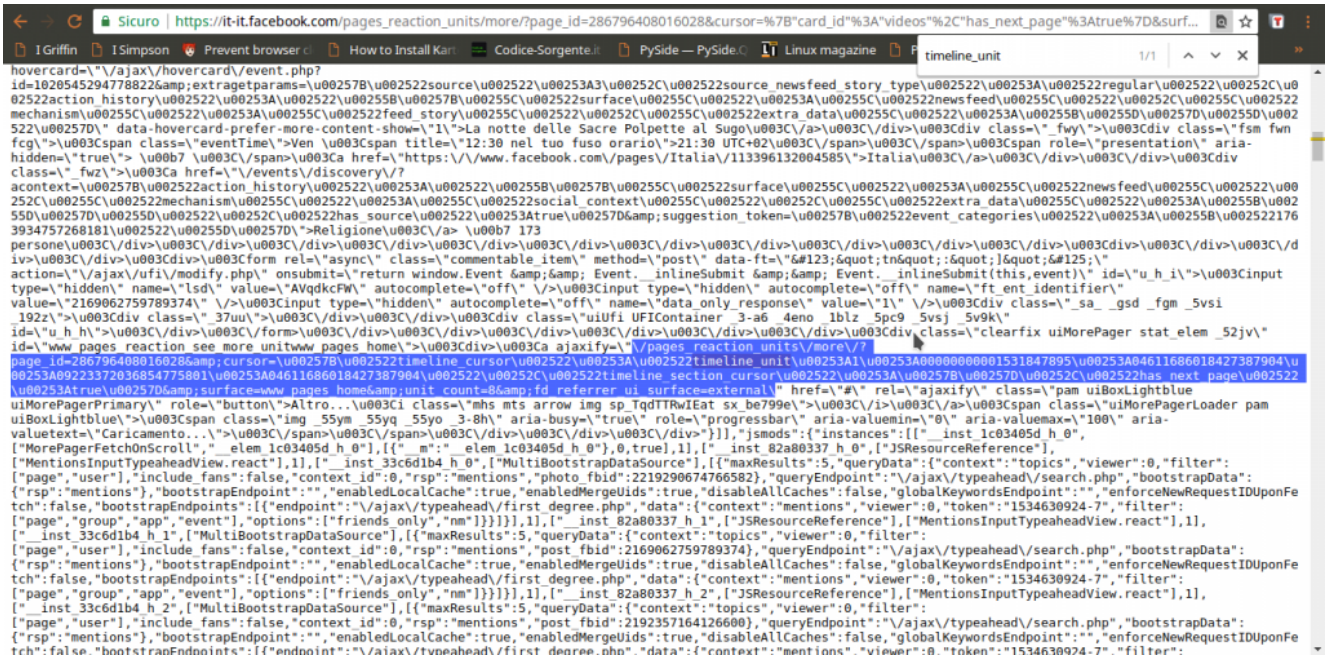
trovato ciò che ci interessava: le altre eventuali richieste sono tutte relative a file accessori, come le immagini.



L’indirizzo della pagina contiene una serie di informazioni importanti, in particolare l’ID della pagina, ed è l’unica richiesta di questo tipo. Possiamo leggere il suo intero URL cliccandoci sopra col tasto destro del mouse e scegliendo **Copy link address**. Aprendo il link, si può capire che forma abbia la risposta: è una sorta di array JSON, una lista di oggetti vari, tra i quali il codice HTML necessario a presentare i post che sono stati richiesti. Si può facilmente distinguere il testo dei post in mezzo a tutto il codice. Alcuni caratteri vengono codificati come Unicode, inclusi alcuni pezzi dei tag HTML, e c’è sempre l’escape per i simboli /, che appaiono come \/, quindi è importante ricordarsi di convertirli in caratteri veri e propri, per poterli riconoscere facilmente (per esempio, \u003C\/p> è in realtà

).

Guardando meglio la risposta di **pages_reaction_units** si scoprono alcune cose interessanti. Innanzitutto, tutti i post che si sono ottenuti vengono presentati tra il testo **{“__html”:** e il testo **]]}**, quindi possiamo selezionarlo facilmente. Inoltre, ogni post è preceduto dalla sua data, in vari formati. In particolare, c’è la forma Unix Time, che è molto comoda da gestire ed è sempre identificata dalla dicitura **data-utime**, quindi potremo distinguere i vari post dividendo l’intero HTML in più pezzi dopo ogni occorrenza della parola **data-utime**. E non solo: si può anche capire come ottenere ulteriori post, facendo un’altra richiesta a questa stessa pagina. Se infatti cerchiamo **pages_reaction_units** all’interno della risposta possiamo notare che c’è l’intero indirizzo di una richiesta come quella che abbiamo appena inviato, ma contenente anche il blocco **timeline_cursor**, con i riferimenti della timeline di Facebook relativi ai prossimi post. Possiamo quindi facilmente estrarre questi riferimenti per confezionare la nostra prossima richiesta.



Ora, tutto quello che è rimasto da scoprire è come costruire l'indirizzo da contattare per ottenere i vari post: sappiamo che serve **pages_reaction_units** più una serie di argomenti (che vediamo nella richiesta estrapolata dal browser). Non tutti gli argomenti sono però necessari, e possiamo scoprire quali siano superflui semplicemente provando a cancellarli uno alla volta e vedendo in quali casi si ottiene comunque il risultato desiderato. Scopriamo quindi che l'indirizzo necessario è qualcosa del tipo: **https://it-it.facebook.com/pages_reaction_units/more/?page_id=286796408016028&cursor={"timeline_cursor":"timeline_unit:1:000000000015286240 41:04611686018427387904:09223372036854775793:04611686018427387904","timeline_section_cursor":{},"has_next_page":true}&surface=www_pages_home&unit_count=8&dpr=1&__user=0&__a=1**. Di questo indirizzo, abbiamo capito che la **timeline_unit** può essere scoperta all'interno di una richiesta precedente, mentre per scoprire l'ID della pagina Facebook basta scorrere il codice HTML della pagina stessa (che si può vedere nel browser Chrome con il prefisso **view-source:**) e cercare proprio **pages_reaction_units**, e subito dopo la parola **page_id**. Giocando un po' con **unit_count** scopriamo che per la prima richiesta possiamo ottenere fino a 300 post, mentre per tutte le successive (quelle in cui si specifica la **timeline_unit**) il massimo che

si può chiedere è 8 post. Tutto il resto è solo un insieme di argomenti vari sempre uguali, che nel nostro programma potremo quindi memorizzare sotto forma di variabili.

La procedura per scaricare tutti i post sarà quindi intuitiva: si accede alla pagina leggendo il suo codice HTML per scoprire l'ID. Con questo si forma il primo URL da contattare per ottenere gli ultimi post. All'interno della risposta si prendono i post dividendoli e salvandoli separatamente, e si cercano anche i riferimenti della timeline_unit per poter fare una nuova richiesta e ottenere altri post, più vecchi. Poi si ripetono continuamente gli ultimi passaggi, leggendo la risposta, salvando i post, costruendo il nuovo indirizzo, e facendo una nuova richiesta, finché Facebook non fornisce più alcun post (il che significa che siamo arrivati all'inizio della pagina e i post sono finiti). Ora dobbiamo tradurre questa idea in uno script Python.

Leggere le pagine web

Cominciamo lo script, tutto in un unico file che chiamiamo **scrapefb.py**:

L’inizio è dato dalla shebang (#!), che su sistemi Unix è utile per automatizzare l’avvio dello script trattandolo come un eseguibile. Poi si devono importare tutte le librerie necessarie: urllib permette di scaricare il contenuto degli URL, e socket permette di stabilire un timeout sulle connessioni per chiuderle se sono inattive. Per fare il parsing della pagina web, cioè per leggere il suo contenuto distinguendo i vari “pezzi”, utilizziamo le espressioni regolari con la libreria re. Abbiamo anche bisogno di lavorare con data e ora, e accedere a funzioni relative al sistema operativo (per lettura e scrittura dei file).

Definiamo uno user agent: si tratta di una semplice stringa di testo che ogni sito richiede a chi vuole ricevere le pagine web, per capire di chi si tratti. Siccome possiamo scriverla come vogliamo, nessuno controlla davvero che stiamo dicendo la verità, possiamo scriverla in modo da convincere Facebook che il nostro script è in realtà il browser web Mozilla Firefox.

Definiamo una funzione che ci aiuti a scaricare tutto il contenuto di una pagina web, e la chiamiamo **geturl**. Prima di tutto, specifichiamo che ci serve lo useragent che abbiamo dichiarato nella sezione globale dello script. Poi ci assicuriamo di non procedere se l’url fornito alla funzione è vuoto, così evitiamo errori inutili. Costruiamo la richiesta HTTP utilizzando l’url. Servono anche un array di dati, che però in questo caso non è necessario visto che non abbiamo un form HTML da fornire alla pagina, e una intestazione. L’intestazione viene costruita con lo user agent, così Facebook ci scambierà per un browser web e non bloccherà la richiesta.

La richiesta HTTP può essere inviata usando la famosa funzione urlopen, e impostiamo anche un timeout. Il timeout è utile per non rimanere bloccati in eterno nel caso la connessione dovesse essere troppo lenta. Con un tempo di 300 secondi,

sappiamo che al massimo dopo 5 minuti la situazione verrà sbloccata. La risposta del server alla nostra richiesta può essere letta con la funzione **read**, e nel caso qualcosa non abbia funzionato impostiamo la risposta (variabile **ft**) come vuota.

In teoria potremmo tenere la risposta così com'è, ma non è una buona idea: il web è una giungla di codifiche, e se non gestiamo la cosa rischiamo di ottenere testi illeggibili. Soprattutto per Facebook, che spesso codifica le varie emoticon sotto forma di caratteri speciali Unicode. Cerchiamo quindi prima di tutto di capire se il server ci suggerisca la codifica della pagina che ci sta inviando. In caso negativo, proviamo a decodificare il testo con la classica codepage di Windows 1252, uno standard sui sistemi Microsoft precedenti a Windows 10. Se non dovesse funzionare, proviamo a decodificare tutti i caratteri usando l'utf-8 togliendo però gli slash inutili (che spesso i server web forniscono per facilitare i browser), e altrimenti cerchiamo di tradurre direttamente l'intera pagina in una stringa python. Comunque sia andata, quindi, avremo una più o meno corretta stringa python piena di tutto il codice html della pagina. Per leggere meglio il suo contenuto, utilizziamo la funzione **html.unescape** per decodificare anche le varie entità dell'html (per esempio, &gt e &lt sono rispettivamente > e <, preziosi per interpretare il codice). L'unescape delle entità html non è fondamentale, ma rende il nostro lavoro più comodo.

Cercare l'ID della pagina Facebook

Cominciamo a scrivere la funzione vera e propria per lo scraping delle pagine di Facebook. La funzione richiede, come argomenti, l'indirizzo della pagina da scaricare, la cartella in cui salvare il risultato, e se si debba salvare il

risultato come tabella CSV invece che come testo TXT.

Innanzitutto ci sono un paio di informazioni, che possiamo memorizzare in alcune variabili. Potremmo anche scriverle direttamente nelle funzioni che le usano, come vedremo, ma tenendole nelle variabili è molto più facile modificarle in futuro se dovesse essere necessario a causa di modifiche nel funzionamento di Facebook. La variabile TOSELECT_FB contiene la stringa da cercare dentro la pagina Facebook per conoscere l'URL che fornisce i vari post. Le due successive variabili sono le stringhe che delimitano l'inizio e la fine dei post nella risposta. Infatti, Facebook non fornisce solo l'elenco dei post della pagina, ma anche una serie di altre informazioni che non ci servono. Per non complicarsi la vita, bisogna avere un output pulito, quindi toglieremo tutto ciò che non ci serve isolando solo il testo presente tra quei due delimitatori. Stabiliamo poi il numero di risultati che vogliamo: il massimo consentito da Facebook (al momento) per la prima richiesta è di 300 post. Inoltre, specifichiamo un periodo di attesa prima di inviare le richieste successive, per evitare che il server possa accorgersi che ne stiamo facendo troppe tutte assieme. Le ultime due rappresentano l'inizio e la fine del link per ottenere i vari post: vedremo tra un po' come costruirlo nella sua interezza.

In questo momento siamo pronti per eseguire la prima richiesta e scaricare la pagina Facebook. L'indirizzo che contattiamo è qualcosa del tipo **https://it-it.facebook.com/chiesapastafarianaitaliana/**. Ovviamente otteniamo soltanto gli ultimi post, proprio quello che un utente normale vede quando carica la pagina. Di per se i post che appaiono non ci interessano, li otterremo contattando direttamente l'URL che fornisce tutti i post. Il codice HTML di questa pagina ci interessa soltanto perché possiamo estrarre delle informazioni. In particolare, vogliamo scoprire il dominio di Facebook, cioè tutto quello che è

compreso tra **https://** e il primo / successivo. Nel caso in esempio è **it-it.facebook.com**, ovviamente è diverso per ogni paese (una pagina spagnola non inizierà con it-it). Cerchiamo anche di capire il nome della pagina, che è tutto ciò che segue il dominio: siccome lo useremo come nome del file in cui salvare i risultati è fondamentale che non ci siano caratteri strani. Usando una espressione regolare, cancelliamo (sostituiamo con "") tutti i caratteri che non siano lettere o numeri. Fondamentale per poter proseguire è il **pageid**, cioè il numero identificativo della pagina che vogliamo scaricare: questa informazione si può recuperare dalla pagina stessa perché è sempre presente in essa un link che contiene tale numero. Il link in questione ha la forma **?page_id=286796408016028&cursor**, quindi possiamo scoprire l'ID cercando ciò che segue la parola **page_id=** e arriva fino al simbolo **&**. Ci si potrebbe chiedere come mai per cercare i vari delimitatori utilizziamo direttamente la funzione index, molto pratica e veloce, mentre per cercare la posizione del **'pages_reaction_units'**, che determina l'inizio del link in cui troviamo la pageid, usiamo le RegEx. La risposta è semplice: per ora trovare questa stringa è facile, ma in futuro potrebbe essere necessario usare una espressione regolare. In questo modo, lo script è già pronto per future modifiche.

Ora che abbiamo tutte le informazioni necessarie, possiamo costruire il nome del file in cui andremo a scrivere i post recuperati. Il nome è dato dalla cartella in cui salvare i file più **fb_** e il nome della pagina. Ovviamente, se l'utente vuole un TXT l'estensione del file sarà TXT, e se vuole un CSV l'estensione sarà CSV. Creiamo anche un altro file, con stesso nome la estensione **.tmp**. Questo è il file in cui andremo ad inserire i vari link già visitati, così se si deve riprendere lo scaricamento dei post di una pagina Facebook non lo si ricomincia da capo ogni volta, ma si riprende da dove ci si era interrotti. Per l'appunto, nel caso il file esista già vuol dire che non si deve ricominciare da capo, quindi si

carica l'intero contenuto del file in una lista, chiamata **alllinks**. In questa lista ogni elemento è un link, perché il file è stato diviso riga per riga (e quando lo scriveremo, metteremo un link in ogni riga). Definiamo anche una variabile che faccia da contatore, per sapere quante richieste di post siano state fatte, e una che stabilisca se stiamo ripristinando un download interrotto o se dobbiamo ricominciare da capo.

Richiedere i post della pagina al server di Facebook

Siamo al momento della raccolta vera e propria dei post della pagina. Siccome dobbiamo fare tante richieste una dopo l'altra, utilizziamo un ciclo. Il ciclo **while** andrà avanti finché la variabile **active** sarà True. Ne consegue che per fermare il ciclo, se necessario, non dovremo fare altro che porre tale variabile uguale a False.

Il link viene costruito unendo il dominio di Facebook, la parte iniziale del link, l'id della pagina, e la parte finale. Sarà quindi qualcosa del tipo **https://it-it.facebook.com/pages_reaction_units/more/?page_id=286796408016028&cursor={"card_id":"videos","has_next_page":true}
&surface=www_pages_home&unit_count=300&referrer&dpr=1&__user=0&__a=1**, come si può notare ci sono tutti i vari pezzi che abbiamo costruito finora. Se provate ad aprire questo indirizzo col browser vi accorgerete che fornisce una serie di informazioni, tra cui l'html dei vari post che sono stati richiesti (cioè gli ultimi 300 post della pagina). Inseriamo il link appena costruito nel file che li deve memorizzare, così se lo script dovesse bloccarsi mentre cercare di recuperare i post sapremo di dover ricominciare da questo preciso link, e non doverli rifare tutti da capo. Usando la

modalità di accesso al file “a” eseguiamo un “append”, cioè inseriamo direttamente questo link alla fine del file, in una nuova riga, senza bisogno di preoccuparci di quali altri link ci fossero prima (non dobbiamo quindi aprire il file, leggerlo, aggiungere il nuovo link, e poi salvarlo). È un risparmio di risorse importante.

Sempre utilizzando la funzione geturl possiamo recuperare anche con il nostro script tutta la risposta del server di Facebook. Siccome ci interessa soltanto la parte con i vari post che abbiamo richiesto, la estraiamo e la memorizziamo nella variabile **postshtml**. Il codice HTML dei vari post va un po’ ripulito: Facebook usa molti caratteri che non sono UTF-8 per gestire le emoticon, in genere sono utf-16. Però per il nostro scopo sono fastidiosi, le emoticon non ci interessano affatto e l’elaborazione dei testi è molto più facile con l’utf-8. Quindi ci assicuriamo di tradurre tutti i caratteri in UTF-8, togliendo anche l’escape dei caratteri speciali. Facebook, infatti, decide che alcuni caratteri sono particolari e li presenta con al loro notazione Unicode, una cosa del tipo \u0001. Questo è molto scomodo per noi, quindi forziamo la trasformazione in caratteri leggibili. A questo punto potrebbero essere rimasti dei simboli che UTF-8 non è in grado di gestire, perché si tratta delle famigerate emoticon UTF-16. Si riconoscono perché il codice Unicode è compreso tra \uD800 e \uDFFF. Siccome non ci interessano, usiamo una semplice espressione regolare per cancellarli, sostituendoli con la stringa vuota “”. Ora abbiamo finalmente l’intero codice HTML dei post, pulito e pronto per essere letto e interpretato. Siccome ogni post di Facebook è contrassegnato da un orario nel formato Unix Time (uno standard di internet), possiamo spezzare il contenuto dell’HTML nei singoli post dividendo proprio in base a **‘data-utime’**, che è la stringa che Facebook usa per indicare l’orario di un post.

In questo momento, la lista **postsarray** contiene i vari post:

in realtà, il primo elemento della lista non contiene post, perché ha tutto l'HTML precedente. Comunque, possiamo scorrere la lista e individuare i post banalmente cercando il loro timestamp, cioè l'orario della pubblicazione. È facile da identificare, perché come dicevamo ogni post viene preceduto da una span (elemento HTML) che contiene una dicitura di questo tipo: **data-utime=\"1531306802\" data-shorten=\"1\" class=\"_5ptz\">**. Siccome noi stiamo dividendo l'HTML a ogni "data-utime", è ovvio che ogni post inizierà con**=\"1531306802\" data-shorten=\"1\"…**, e quindi l'orario in formato Unix sarà il primo numero tra virgolette (nell'esempio è **1531306802**). Per essere sicuri di nona vere problemi, usiamo una RegEx per cancellare dal timestamp ottenuto qualsiasi cosa non sia un numero, e convertiamo il risultato in un **int**, cioè un numero intero. Nel caso non sia possibile risalire a questo numero, come per il primo elemento della lista che non è un vero post, consideriamo il numero pari a zero. Poi, usando datetime, possiamo convertire questo timestamp in una data facilmente leggibile, nel formato anno-mese-giorno ore:minuti:secondi. La data viene quindi aggiunta alla lista **timearray**, che abbiamo appositamente creato. Ciò significa che per ogni elemento di **postsarray**, cioè ogni post della pagina Facebook, abbiamo un corrispondente elemento di **timearray**, cioè la data della pubblicazione del post stesso.

Tutto il testo (se c'è) del post numero **i** si trova dentro all'elemento **postsarray[i]**, ma è ovviamente circondato da un sacco di altri pezzi di HTML che non ci servono. Per estrapolare soltanto il testo dei post basta prelevare tutto ciò che si trova all'interno dei paragrafi (che nella risposta di Facebook sono i tag

<\/p>). Bisogna ricordare che nello scrivere l'espressione regolare per trovare i tag il carattere \ ha bisogno di una sequenza di escape lunga, e va scritto come \\\\. La funzione **finditer** crea l'array **indexes**, che contiene tutte le posizioni in cui si trovano i vari paragrafi: un post di Facebook può

infatti essere diviso in tanti paragrafi, e noi li vogliamo tutti. Ciascun elemento di **indexes**, contiene in realtà due informazioni: la prima (cioè **0**) è l'inizio del paragrafo, e la seconda (cioè **1**) è la fine del paragrafo. Usando il classico sistema di slicing delle stringhe di Python, si può banalmente estrarre il testo di ogni paragrafo semplicemente partendo dal carattere iniziale e finale (quindi **postsarray[i][start:end]**, perché la stringa è **postsarray[i]**). Alla fine del ciclo for che legge tutti i vari **indexes**, avremo la variabile **thispost** che contiene tutti i vari paragrafi uniti, senza gli altri tag inutili.

Possiamo assegnare tutto il testo del paragrafo all'elemento stesso da cui eravamo partiti, così lo avremo ripulito. Prima, però, togliamo i tag che ancora esistono. Per esempio, il grassetto viene realizzato con i tag , quindi noi cancelliamo tutto cioè che si trova tra i simboli **<** e **>**. E cancelliamo anche gli slash non necessari. Quindi, **gatti\/cani** diventa **gatti/cani**. Alla fine presentiamo l'array sul terminale, così è più facile fare il debug e capire se qualcosa non vada bene. Lo scraping è pur sempre legato a qualcosa di molto casuale, e può capitare che in situazioni particolari qualcosa improvvisamente non funzioni.

Scoprire gli ID dei prossimi post da scaricare

Ora abbiamo ricostruito la lista **postsarray**, che contiene tutti i post presenti nell'attuale risposta di Facebook. Dobbiamo ancora capire come costruire la prossima richiesta, per ottenere una nuova risposta.

Le varie richieste che vengono inviate sono qualcosa del tipo **https://it-it.facebook.com/pages_reaction_units/more/?page_id=**

286796408016028&cursor={“timeline_cursor”:”timeline_unit:1:000 00000001528624041: 04611686018427387904:09223372036854775793: 04611686018427387904″,”timeline_section_cursor”:{}, “has_next_page”:true}&surface=www_pages_home &unit_count=8&dpr=1&__user=0&__a=1. Se ci si fa caso, è praticamente identica alla prima richiesta, con due differenze fondamentali: l’argomento **cursor** contiene i riferimenti della timeline di Facebook che indica da dove iniziano i post da scaricare. E poi c’è la **unit_count** che è limitata a **8**, quindi si possono scaricare al massimo 8 post per volta. Siccome lo stesso Facebook ha bisogno di sapere quali siano i riferimenti della timeline della pagina da scaricare, è ovvio che nella attuale risposta (quella che abbiamo appena ricevuto) ci debbano essere. E infatti ci sono, si possono trovare proprio nella forma dell’url con i due argomenti **cursor** e **unit_count**, quindi possiamo ottenerli cercando questi pezzi dell’URL dentro la stringa **newhtml** (che contiene l’ultima risposta che abbiamo ottenuto da Facebook). Siccome la prima parte dell’url è sempre la stessa, non dobbiamo fare altro che modificare la parte finale includendo i riferimenti della timeline appena ottenuti nella variabile **lending**. In questo modo, al prossimo ciclo verrà di nuovo costruito il **link**, ma usando questo **lending** come parte finale, e si potrà fare la nuova richiesta a Facebook per i successivi 8 post della pagina. Ovviamente, il testo della timeline estrapolato va un po’ pulito, con le funzioni che avevamo già visto per la rimozione dei caratteri Unicode inutili e per la decodifica dell’url. Se non riusciamo a trovare un url per i prossimi post, vuol dire che sono finiti, quindi dobbiamo interrompere il ciclo impostando la variabile **active** come falsa.

Se per qualche motivo non è stato possibile recuperare i post e le date dei post, le due apposite liste vengono inizializzate come vuote, così il programma non si bloccherà.

È arrivato il momento di salvare il risultato in un file.

L’elenco dei post scaricati durante questo ciclo è nella lista **postsarray**, possiamo trasformarla in un testo da salvare nel file prendendo i vari elementi della lista e aggiungendoli alla variabile **postsfile**, uno in ogni riga (**\n** indica un invio a capo riga). Se si desidera che il file sia un CSV, il testo del post viene preceduto dalla data di pubblicazione del post, che si trova nella lista timearray. La data e il testo del post sono separati da una tabulazione, cioè **\t**, perché se utilizzassimo altri simboli come virgola e punto virgola il risultato sarebbe inaffidabile: un post di Facebook può facilmente contenere della punteggiatura, ma non una tabulazione.

Ora che il testo da scrivere nel file è tutto nella variabile **postsfile**, dobbiamo capire se sia necessario creare il file da capo oppure no. Se questa è la prima iterazione del ciclo, e non si sta eseguendo il ripristino di un download interrotto precedentemente, bisogna scrivere il testo nel file, dunque sovrascrivendo qualsiasi cosa ci fosse (se il file esisteva già). Altrimenti, bisogna soltanto aggiungere l’attuale testo a ciò che era già stato scaricato in precedenza, usando per la modalità di scrittura append (cioè **a**) che avevamo già visto.

Ovviamente, alla fine del ciclo si incrementa di 1 il contatore **timelineiter**, che ha per l’appunto la funzione di farci sapere quante iterazioni sta facendo il programma alla ricerca di altri post da scaricare. Inoltre, prima di concludere le operazioni del ciclo, e ricominciarlo da capo, attendiamo un certo numero di secondi. Questo è importante perché se riprendessimo immediatamente a fare richieste a Facebook, il server potrebbe accorgersi che stiamo insistendo troppo e magari bloccare la connessione.

Il blocco principale dello script

Terminata la funzione per lo scraping di Facebook, ricomincia il blocco principale dello script. Per sicurezza, controlliamo che in questo momento sia stata specificamente richiesta, dall'interprete Python, la funzione main. Questo avviene soltanto se lo script è stato lanciato direttamente dal terminale, e non se è stato importato in un altro script. Questo controllo facilita un eventuale utilizzo del nostro script come libreria per altri programmi.

Ora, si definisce la pagina Facebook da cui si parte, che può essere fornita dall'utente come primo argomento dello script. Il secondo argomento, se esiste, deve rappresentare la cartella in cui si vuole ottenere il file di output, e se non è specificato si suppone che sia la cartella attuale (cioè **./**, presumibilmente la stessa in cui si trova anche lo script). Il terzo argomento, se esiste, può essere la parola "CSV": in questo caso, significa che l'utente vuole ottenere il CSV invece del TXT. Le varie informazioni vengono passate alla funzione di scraping per cominciare il download dei vari post. L'utilizzo è quindi molto semplice, e richiede praticamente soltanto l'indirizzo completo della pagina che si vuole scaricare, così come ogni utente può leggerlo in un browser web. Per esempio, si può lanciare lo script col comando **python3 scrapefb.py https://it-it.facebook.com/chiesapastafarianaitaliana/ ./ CSV** in modo da ottenere nella cartella corrente un CSV con data e testo di tutti i post della pagina della Chiesa Pastafariana Italiana, oppure si può usare il comando **python3.exe scrapefb.py https://it-it.facebook.com/chiesapastafarianaitaliana/ C:\Temp** per ottenere il TXT nella cartella **C:\Temp**.

Il codice completo

Potete trovare il codice completo dello script all’indirizzo https://gist.github.com/zorbaproject/c1f8fff28cd0becea3a0fb6d0badd159. Per utilizzarlo è necessario avere Python3 installato sul proprio sistema, ed è stato provato sia su GNU/Linux che su Windows.