

AGIW PROGETTO 1

ESTRAZIONE DI DATI DAL WEB

NIGHTCRAWLER:

Patrizio Di Fraia

Ion Chiriac

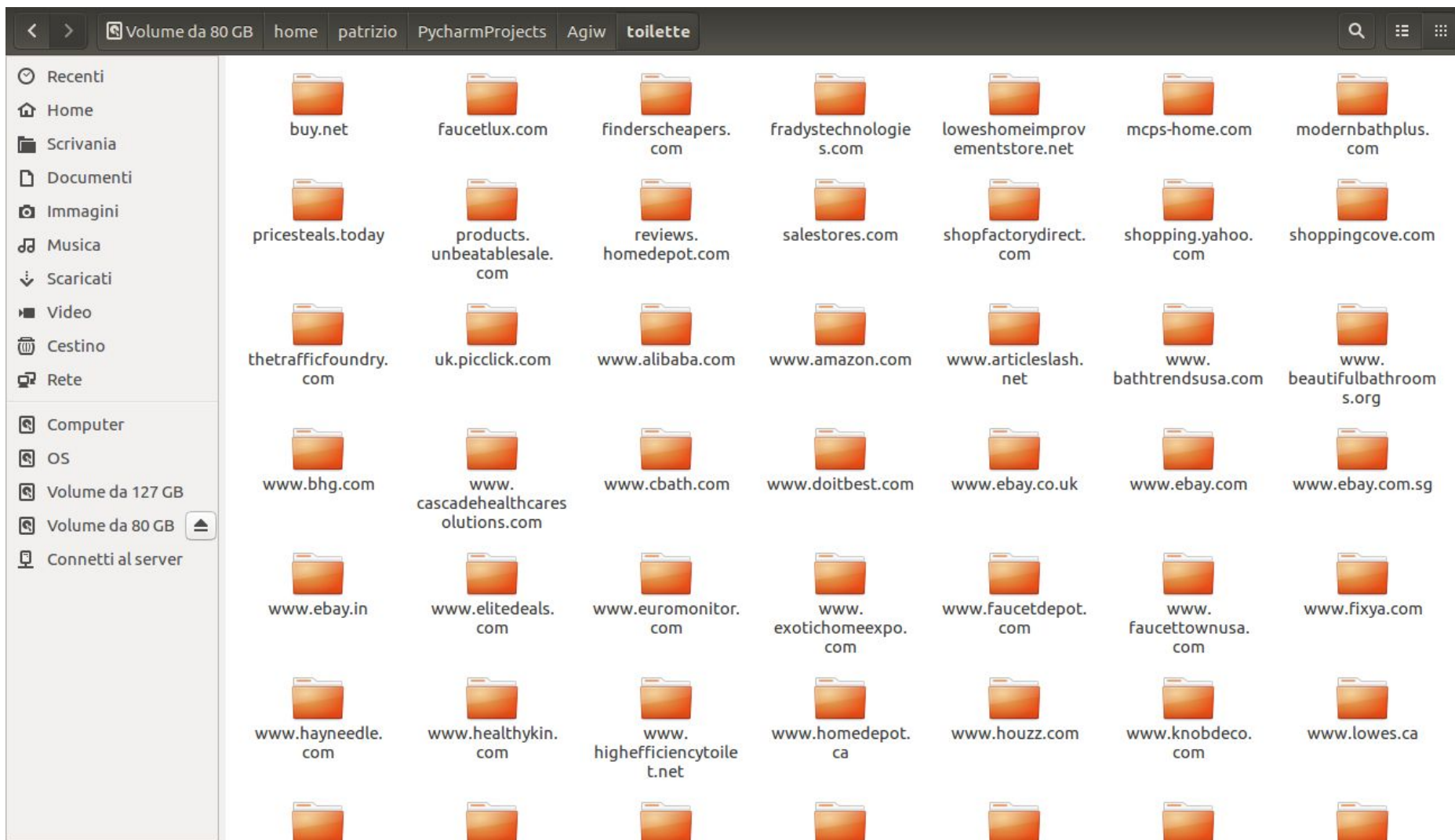
Andrea D'Antonio

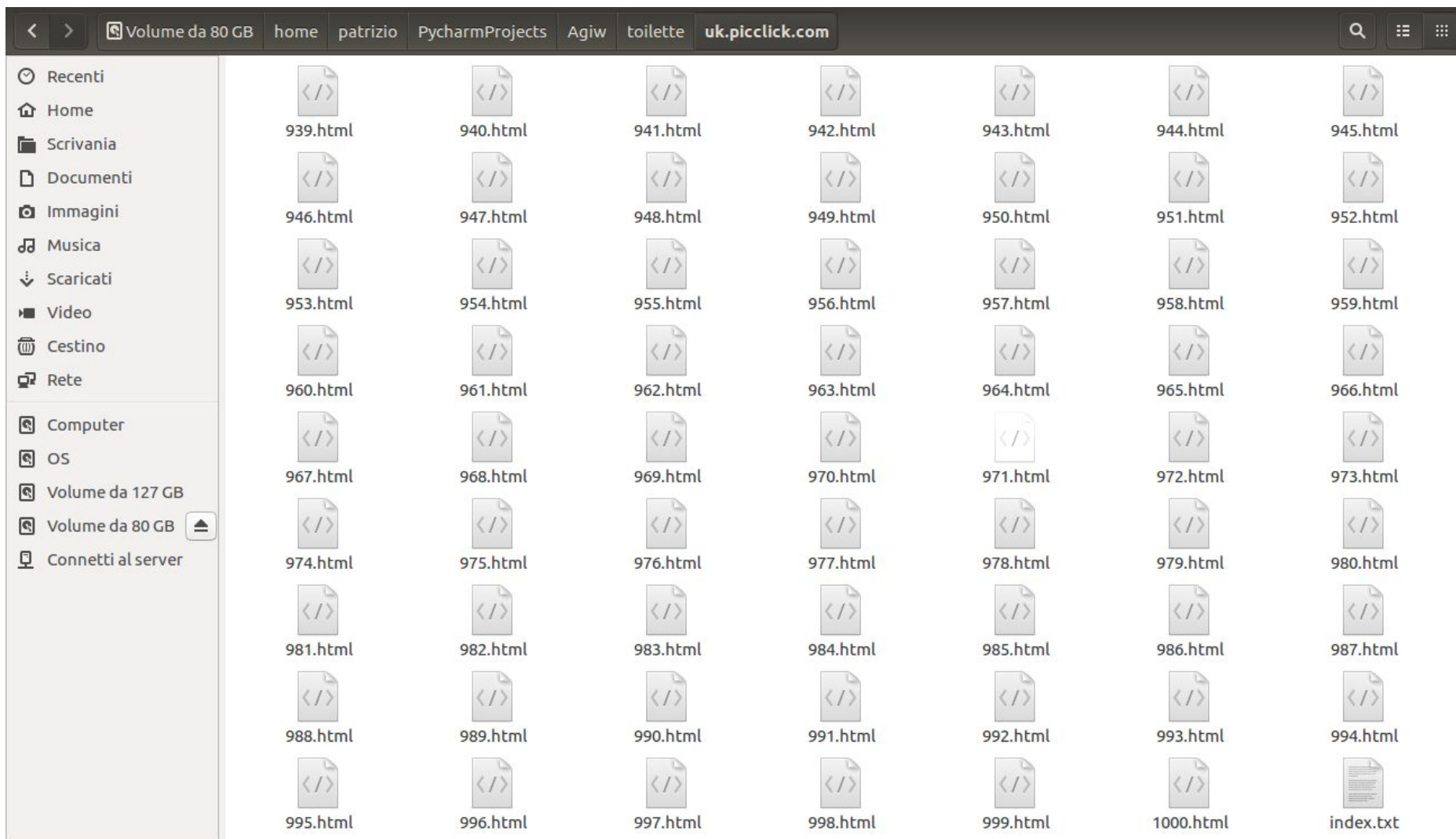
Il dominio applicativo da noi studiato è composto da:


- 40183 url differenti
- 223 siti web differenti
- Prodotti di siti e-commerce, in particolare coltelli e apparecchi sanitari (per le fasi 0, 1, 2)

Step 0

- Partendo da due file json(uno per dominio di interesse) contenente gli url e siti web specificati sopra, ne abbiamo scaricato le relative pagine html
- Questa fase come output prevede una cartella per ogni sito web, all'interno della quale vengono salvati i file progressivo_sito.html e un file txt che associa ad ogni link o il rispettivo file html o il numero dell'errore





```
Apri  Salva
http://elitedeals.com/tot-sn970m-12.html      ./toilette/www.elitedeals.com/1.html
http://elitedeals.com/tot-cst744sldb-11.html  ./toilette/www.elitedeals.com/2.html
http://elitedeals.com/tot-cst744slb-11.html   ./toilette/www.elitedeals.com/3.html
http://elitedeals.com/tot-cst744eg-01.html    ./toilette/www.elitedeals.com/4.html
http://elitedeals.com/tot-cst423sf-01.html    ./toilette/www.elitedeals.com/5.html
http://elitedeals.com/tot-cst424ef-01.html    404
http://elitedeals.com/tot-cst423sfg-01.html   ./toilette/www.elitedeals.com/7.html
http://elitedeals.com/tot-ms624214cefg-11.html ./toilette/www.elitedeals.com/8.html
http://elitedeals.com/tot-cst716db-01.html    404
http://elitedeals.com/tot-ce705lg-1ln-534-01.html ./toilette/www.elitedeals.com/10.html
```

Testo semplice ▾ Larg. tab.: 8 ▾ Rg 1, Col 1 ▾ INS

- Per ottenere questo output abbiamo utilizzato le librerie python aiohttp e asyncio, che consentono di eseguire richieste in modo asincrono (si passa dalle 18h dello script sincrónico a 30 minuti di questo). L'output è di 8101 link validi.

```

import aiohttp
import asyncio
import asyncio_timeout
import os
import json
import datetime
from collections import OrderedDict
import sys

async def download_coroutine(session, url, folder, resultfile, progressive):
    with asyncio_timeout.timeout(30):
        async with session.get(url) as response:
            time = datetime.datetime.now().strftime("%A, %d. %B %Y
%H:%M:%S.%f")[:-3]
            filename = os.path.basename(str(progressive) + ".html")
            output = os.path.join(folder, filename)
            if str(response.status)[0] == "2":
                with open(output, 'wb') as f_handle:
                    while True:
                        chunk = await response.content.read(1024)
                        if not chunk:
                            break
                        f_handle.write(chunk)
                    with open(resultfile, "a") as resultfile:
                        resultfile.write(url + "\t" + output + "\n")
                        print(url + "\t" + output + " [" + time + "]")
            else:
                with open(resultfile, "a") as resultfile:
                    resultfile.write(url + "\t" + str(response.status) + "\n")

```

```

        print(url + "\t" + str(response.status) + " [" + time + "]")
        return await response.release()
    async def main(loop):
        data = json.load(open(sys.argv[1]), object_pairs_hook=OrderedDict)
        folderpath = "." + sys.argv[2]

        for key, value in data.items():
            sitepath = os.path.join(folderpath, key)
            resultfile = os.path.join(sitepath, "index.txt")
            if not os.path.exists(sitepath):
                os.makedirs(sitepath)
            async with aiohttp.ClientSession(loop=loop) as session:
                progressive = 1
                tasks = []
                for url in value:
                    tasks.append(download_coroutine(session, url, sitepath,
resultfile, progressive))
                    progressive = progressive+1
                try:
                    await asyncio.gather(*tasks)
                except Exception as e:
                    time = datetime.datetime.now().strftime("%A, %d. %B %Y
%H:%M:%S.%f")[:-3]
                    data = str(e)
                    info = data[:25] + (data[25:] and '..')
                    print("other error: " + info + " [" + time + "]")
                    pass
if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))

```

Step 1

- Testare l'estrattore SPEXA sulle pagine html scaricate
- Il codice richiede come parametro l'indirizzo http del sito e non la pagina html, abbiamo quindi replicato lo script della fase 1 aggiungendo una riga in python che consente di lanciare comandi bash

```
bashCommand = "python2.7 -m src.model.specificationextractor %s %s" %  
(str(url), folder2+"/"+str(progressive))  
process = subprocess.Popen(bashCommand.split(), stdout=subprocess.PIPE)  
output, error = process.communicate()
```

- Sugli 8101 url validi
 - 1124 (13 %) danno in output un file json
 - 792 (70% degli estratti e 9% dei totali) sono quelli non vuoti.

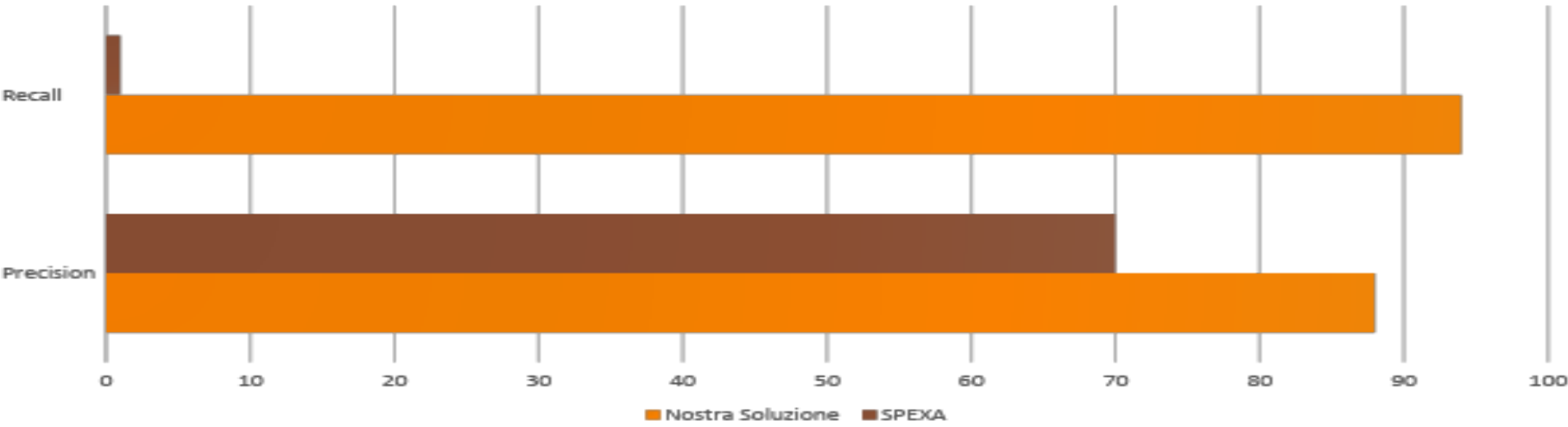
Step 2

- Lo step 2 prevede di cercare il modo di battere SPEXA
- Per prima cosa abbiamo cercato di capire chi fosse il nostro avversario
 - 13% di url con output
 - 70% dei quali con estrazione
 - 9% totale
 - Date le basse performance ci siamo messi nel caso peggiore in cui tutti e 792 i json estratti effettivamente contenevano cose sensate e utili (in realtà non è così), pensando che se realizziamo un sistema che batte tali prestazioni sicuramente lo batte in quelle reali.
- Inizialmente in modo furbesco abbiamo pensato a due soluzioni
 - fare manualmente l'estrazione di 792 siti
 - poco divertente
 - prendere il codice di SPEXA e migliorarlo così da batterlo sicuramente
 - poca fantasia


- La nostra soluzione ha preso spunto dalle tecniche viste a lezione (per esempio Vertex, Weir), ovvero sfruttiamo il fatto che pagine dello stesso sito sono simili e aggiungiamo un lavoro manuale per le annotazioni
 - Scrapely, libreria di Python che consente di addestrare dei modelli per riuscire ad estrarre informazioni dal testo non strutturato.
 - Per l'addestramento necessita di un file di annotazioni (json) e di un file su cui testarlo(url http, pagina html)
 - Manualmente abbiamo realizzato il file di annotazioni per ogni sito web (mediamente 30 all'ora)
- Risultati:
 - 7631 estrazioni su 8101 url (94,2%)
 - 6749 contenente informazioni(88,4% sugli estratti, 83,3% sui totali)

Grafico e tabella della presentazione 16/04




	JSON ESTRATTI	JSON VALIDI	RECALL	PRECISION
NOSTRA SOLUZIONE	7631	6749	0.942	0.884
SPEXA	1124	792	Circa 0.1	0.704



Esempio Funzionamento dell'Algoritmo



Mouse over image to zoom



SET OF 6 RESTAURANT STEAK KNIVES WOOD HANDLES USE TO CUT BREAD

★ ★ ★ ★ ★ 1 product rating

Condition: **New**

Quantity: 525 available / 475 sold

Price: **US \$3.69**
Approximately C \$7.77

[Buy It Now](#)

[Add to cart](#)

[Add to watch list](#)

475 sold 50 watchers Longtime member

Shipping: **US \$5.99 (approx. C \$7.77)** Standard International Shipping
[See details about international shipping here.](#)

Delivery: Estimated Delivery within 11-19 business days
Seller ships within 5 days after receiving cleared payment

Payments: [PayPal](#) [VISA](#) [MasterCard](#) [American Express](#) [Discover](#)
Credit cards processed by PayPal
[See payment information](#)

Returns: 14 day returns, buyer pays return shipping
Guarantee: **ebay** MONEY BACK GUARANTEE
Get the item you ordered or get your money back
Covers your purchase price and original shipping

[Add to watch list](#)

Top-Rated Seller
[nesly_kitchen_store](#) (2643 ★)
100% Positive feedback

- Consistently receives highest buyers' ratings
- Ships items quickly
- Has earned a track record of excellent service

[Save this Seller](#)
[Contact seller](#)
[Visit store](#)

un file json di train può essere il seguente:

```
{  "Name": "SET OF 6 RESTAURANT STEAK KNIVES WOOD HANDLES USE TO CUT BREAD",  "Condition": "New",  "Price": "US $3.69",  "Item location": "Jerusalem, Israel"}
```

- Sebbene certi delle prestazioni del nostro sistema abbiamo preso e controllato manualmente cosa estraeva SPEXA e cosa il nostro sistema sulle stesse pagine(che sono molte meno delle nostre), abbiamo inoltre verificato su un campione dei siti e i risultati sono quelli che ci aspettavamo
 - Principali problemi sono legati alla presenza di informazioni ricavate da script JavaScript e quindi non si riescono ad ottenere dal testo, e da prodotti non più presenti e che quindi non da errore nel collegamento ma non contiene nulla (avendo già ottime prestazioni non abbiamo indagato oltre lasciandoli tra quelli da cui non abbiamo estratto informazioni).

```
[
  {
    "Condition": [
      "New"
    ],
    "Item location": [
      "GB, United Kingdom"
    ],
    "Name": [
      "Premium Soft Close Toilet Seat in White | Top Fixing Metal Hinges | BRAND NEW | eBay"
    ],
    "Price": [
      "314.95"
    ]
  }
]
```

Project

- agiw ~/PycharmProjects/agiw
 - estratti_fase_1
 - estratti_fase_2
 - fase_3
 - valid_urls
 - .gitignore
 - asyncWebpagesDownload.py
 - dexter_urls_category_camera_2018-0
 - dexter_urls_category_monitor_2018-0
 - dexter_urls_category_software_2018-0
 - Fase1_Spexca.py
 - README.md
 - scrapelyScript.py
 - webpagesDownload.py
 - webpagesValidator.py
- External Libraries
- Scratches and Consoles

1: Project 2: Favorites 3: Structure

```
8 """
9  Use: ./scrapelyScript.py data_file.json train_folder output_folder
10 """
11
12
13 def main():
14     data = json.load(open(sys.argv[1]), object_pairs_hook=OrderedDict)
15     folderpath = "." + sys.argv[3]
16     for key, value in data.items():
17         sitepath = os.path.join(folderpath, key)
18         if not os.path.exists(sitepath):
19             os.makedirs(sitepath)
20         progressive = 1
21         s = Scraper()
22         try:
23             train_data = json.load(open("." + sys.argv[2] + "/" + key + ".json"), object_pairs_hook=OrderedDict)
24             train_url = value[0]
25             s.train(train_url, train_data)
26         except Exception as e:
27             data = str(e)
28             print("other error: " + data)
29             pass
30         for url in value:
31             filename = os.path.basename(str(progressive) + ".json")
32             output = os.path.join(sitepath, filename)
33             try:
34                 result = s.scrape(url)
35                 print(result[:10] + (result[10:] and '..'))
36                 with open(output, 'w') as f_handle:
37                     json.dump(result, f_handle, sort_keys=True, indent=4)
38                 progressive = progressive + 1
39             except Exception as e:
40                 data = str(e)
41                 print("other error: " + data)
42                 pass
43
44
45 main()
46
```

main() > for key,value in data.items() > for url in value

Step 3

- Per lo step 3 l'obiettivo è quello di testare la scalabilità del sistema realizzato nello step 2
- Per farlo abbiamo preso gli url di altri gruppi che erano su domini diversi
- Innanzitutto abbiamo fatto operazioni di preprocessing sul sistema realizzato
 - Abbiamo salvato il modello poichè è dipendente dal test e quindi se cambia la pagina rischiamo di perdere tutto
 - Abbiamo modificato lo script nel seguente modo
 - se trova un modello addestrato per quell'url utilizza quello
 - altrimenti testa tutti i modelli tra quelli presenti e salva il sito in una lista di quelli da fare
 - si pongono vincoli sulle dimensioni per selezionare il migliore e si eliminano i tag e i separatori così da avere un file più pulito possibile
 - il risultato non è al livello di quella ottenibile con l'aggiunta del modello per quel sito
- miglioria non implementata: matchare il migliore sulla base di keywords, ma dato che comunque è un sistema che va aggiornato, riteniamo che convenga inserire man mano modelli per i vari siti per avere un risultato migliore

```

import sys
import json
import os
from collections import OrderedDict
from scrapely import Scraper
'''
Uso: ./multipleExtractions.py data_file.json template_folder
output_folder
'''

def main():
    data = json.load(open(sys.argv[1]), object_pairs_hook=OrderedDict)
    folderpath = "." + sys.argv[3]
    listfile = os.listdir(sys.argv[2]) #lista di train template
    lista=[]
    for key, value in data.items():
        sitepath = os.path.join(folderpath, key)
        if not os.path.exists(sitepath):
            os.makedirs(sitepath)
        progressive = 1
        second_progressive = 1
        for url in value:
            if os.path.isfile("." + sys.argv[2] + "/" + key + ".json"):
                filename = os.path.basename(str(progressive) + ".json")
                output = os.path.join(sitepath, filename)
                try:
                    s = Scraper().fromfile(open("." + sys.argv[2] + "/" + key +
".json"))
                    result = s.scrape(url)
                    print(result[:10] + (result[10:] and '..'))
                    with open(output, 'w') as f_handle:
                        json.dump(result, f_handle, sort_keys=True, indent=4)
                    progressive = progressive + 1

```

```

except Exception as e:
    data = str(e)
    print("other error: " + data)
    pass
else:
    if key not in lista:
        lista.append(key)
    for file in listfile:
        filename =
os.path.basename(str(progressive)+"_"+str(second_progressive)+".json"
)

        output = os.path.join(sitepath, filename)
        try:
            s = Scraper().fromfile(open(sys.argv[2] + "/" + file))
            result = s.scrape(url)

            temp = str(result)
            if 10 < len(temp) < 3000:
                data = json.loads(temp.replace('""', ''))

            print(result[:10] + (result[10:] and '..'))
            with open(output, 'w') as f_handle:
                json.dump(data, f_handle, sort_keys=True, indent=4)
            second_progressive = second_progressive + 1
        except Exception as e:
            data = str(e)
            print("other error: " + data)
            pass

        progressive = progressive + 1
    dafare=open("dafare.txt","a")
    for i in lista:
        dafare.writelines(i + "\n")
main(

```

```
[
  {
    "Color": [
      "Uncut"
    ],
    "Name": [
      "Battlefield : Bad Company 2 Vietnam Sealed / scanned cdkey"
    ],
    "Place of Origin": [
      "Battlefield: Bad Company 2 Vietnam"
    ],
    "description": [
      "Battlefield : Bad Company 2 Vietnam Sealed / Scanned Cdkey - Buy Bad Company Vietnam Product"
    ],
    "highPrice": [
      "12.5"
    ],
    "lowPrice": [
      "11"
    ],
    "priceCurrency": [
      "EUR "
    ]
  }
]
```



```
[
  {
    "Name": [
      "strong software of interactive whiteboard"
    ],
    "description": [
      "Strong software of interactive whiteboard in China"
    ]
  }
]
```

Considerazione generale sul sistema

Con questo metodo di lavoro, la nostra soluzione può essere utilizzata per domini completamente nuovi e visto che teniamo traccia dei siti senza template, possiamo periodicamente farli e aggiungerli alla lista rendendo il programma sempre più solido. Inoltre, magari con l'utilizzo del crowdsourcing, le annotazioni dei domini possono essere fatte più velocemente. E' un lavoro adatto ad essere diviso in semplici task che possono essere eseguiti facilmente dagli worker.

Probabilmente con tecniche più sofisticate di ML o IA potrebbero uscire dei risultati migliori senza uso di annotazioni manuali, ma anche sulla base del background accademico ci riteniamo molto soddisfatti dei risultati e delle performance ottenute. Come detto anche a lezione è vero che ML e IA aiutano, ma per funzionare hanno bisogno di dataset per eseguire l'addestramento, e quindi un lavoro manuale da qualcuno in qualche modo va fatto.