

# Relazione Progetto Data and Document Mining

Bernardo Tiezzi

E-mail address

`bernardo.tiezzi@stud.unifi.it`

## Abstract

*La relazione seguente analizza le fasi realizzative del progetto d'esame di Data and Document Mining per il corso di Laurea Magistrale in Ingegneria Informatica dell'Università degli Studi di Firenze. L'idea alla base dello sviluppo è implementare un programma che emuli l'algoritmo OCR Text Recognition, al fine di ottenere una classificazione adeguata di varie pubblicazioni.*

## Future Distribution Permission

The author(s) of this report give permission for this document to be distributed to Unifi-affiliated students taking future courses.

## 1. Introduzione

I sistemi di riconoscimento ottico dei caratteri, detti anche **OCR** (Optical Character Recognition), sono programmi dedicati al rilevamento dei caratteri contenuti in un documento e al loro trasferimento in testo digitale leggibile da una macchina. Di solito tale conversione viene effettuata da uno scanner. Il testo può essere convertito in formato **ASCII** semplice, **UNICODE** o, usufruendo di programmi più avanzati, in formati che tengono di conto l'impaginazione del testo stesso. L'OCR è dunque un campo di ricerca dell'AI legato al riconoscimento delle immagini.

Ad oggi esistono numerosi siti che raccolgono varie tipologie di pubblicazioni testuali digitalizzate: per ogni testo il sito consente all'utente di ottenerne un formato *PDF* oltre a fornirne una propria versione web. La maggioranza dei documenti non presenta tuttavia una classificazione testuale adeguata, che ne semplifichi la comprensione.

L'idea alla base del progetto è dunque ovviare al disagio fornendo un algoritmo di tipo OCR Text Recognition, che possa essere utilizzato su tipologie differenti di documenti. Dato in input un testo digitale, il programma fornirà in output la classificazione concettuale del documento stesso, in formato PDF.

## 2. Obiettivi di progettazione

Per ottenere un programma in grado di eseguire correttamente la classificazione, sono stati ottenuti a-priori documenti in formato pdf, forniti dal sito [SpringerLink](#), assieme ai relativi file *HTML*, forniti dalla piattaforma. L'obiettivo è utilizzare i tag html della pagina web, che riportano la struttura del documento, in modo da individuare una corrispondenza testuale con il documento PDF ed ottenere la corretta classificazione.

## 3. Strumenti di Progettazione

Il progetto è stato implementato utilizzando il linguaggio di programmazione python e l'IDE *Pycharm*. Le librerie principali, impiegate durante lo sviluppo del codice, sono:

- **BeautifulSoup**: libreria utilizzata per leggere i file HTML ed ottenere i blocchi di testo, in base alla tipologia di tag HTML e alla relativa classe di appartenenza.
- **PyMuPDF**: libreria impiegata per eseguire una scansione sui documenti PDF, fornendo in output un nuovo file che riporti i risultati ottenuti dal programma.
- **Scikit-Image**: libreria necessaria per individuare la presenza di immagini all'interno del

documento non rilevabili attraverso l'utilizzo di PyMuPDF.

#### 4. Descrizione dell'elaborato

Il programma è stato implementato come *python project*, suddiviso su più *file.py*. Di seguito sono descritti i file principali, andando ad analizzare le funzionalità di ognuno necessarie all'esecuzione del software.

##### 4.1. main.py

Il file **main.py** è il file principale del programma. Le librerie **BeautifulSoup** e **PyMuPDF** vengono incluse all'interno del sorgente. Inizialmente viene eseguito il download per il file HTML e PDF passando in input al programma l'indirizzo della pagina web (viene eseguito un controllo ogni volta per evitare di ottenere duplicati dei file). La piattaforma **SpringerLink** fornisce una banca dati contenente milioni di documenti, accessibili tramite account Springer o account istituzionale. Per ottenere i file viene utilizzato il server proxy, fornito dall'Università degli Studi di Firenze, assieme alle credenziali previste dalla modalità di autenticazione servizi SIAF.

```
with requests.Session() as s:
    s.proxies = {'https':
        'http://[Username]:[Password]
        @proxyunifi.unifi.it:8888'}

    response_link = s.post(link)
    r_link = s.get(link)
```

Il file HTML relativo al documento viene letto utilizzando la libreria *BeautifulSoup*, che consente di eseguire una ricerca all'interno del documento, filtrando i tag presenti in base alla tipologia e alla classe di appartenenza.

```
soup = BeautifulSoup(f.read(),
    'html.parser')
soup.find_all('type', class_='class')
```

Ogni blocco di testo selezionato consente di definire la struttura del documento. La scelta dei tag è ricaduta sulle seguenti tipologie:

- i tag heading per riconoscere l'intestazione e l'organizzazione del documento ipertestuale.

La scelta si limita alle prime 4 tipologie più frequenti: da *h1* ad *h4*.

- le equazioni e formule matematiche presenti all'interno del file. I documenti analizzati sono per la maggior parte articoli scientifici
- i blocchi contenenti le descrizioni di immagini e tabelle
- i riferimenti bibliografici.

Sfruttando la libreria PyMuPDF si effettuano modifiche al documento PDF.

```
import fitz
doc = fitz.open(f"{path_to_pdf_file}")
for page in doc:
    texts = page.get_text("json")
    texts = json.loads(texts)
    ...
```

Ogni pagina viene suddivisa in blocchi di testo, andando ad eseguire un riscontro con il contenuto di ogni tag selezionato. La classificazione viene eseguita disegnando un riquadro attorno ad ogni blocco di testo; se al termine dei controlli non si evidenzia nessuna appartenenza ad una classe semantica, verrà assegnata un'etichetta generica.

```
for block in texts['blocks']:
    ...
    page.draw_rect
    (Rect(block['bbox']),
    color=color_block, width=1.5)
```

Per individuare i titoli di paragrafi e sottoparagrafi è necessario controllare sia il font-weight del testo (Bold o Italics), sia la somiglianza tra la stringa selezionata nella pagina ed ogni stringa appartenente alla lista delle intestazioni. Se la percentuale di somiglianza risulta superiore ad un valore di soglia (90%), viene definita l'etichetta del blocco e si prosegue con l'iterazione successiva. Per individuare le equazioni il procedimento è analogo alla ricerca degli headers di testo; tuttavia la libreria PyMuPDF potrebbe non essere in grado di riconoscere alcuni simboli matematici ( $\sum$ ,  $\int$ ,  $\pm$  etc.), oltre alle parentesi di sistema, disposte su più colonne. Per ovviare al problema vengono eseguiti ulteriori controlli, dividendo in blocchi l'equazione e confrontando ogni

sequenza generata con il blocco di testo selezionato.

```
k = len(string.replace(" ", ""))
it_eq = 0
if k < len(eq.replace(" ", "")):
    while it_eq + k <= len(eq):
        if block['type'] == 0 and
           SequenceMatcher(None,
                             eq[it_eq:it_eq + k], s.replace(" ", "")).ratio() >= 0.6:
            ...
```

Una volta conclusi i restanti controlli per le classi scelte (descrizione immagini, tabelle e bibliografia), si effettua una verifica su casi particolari:

- il blocco di testo, pur non riconosciuto, è posto sulla stessa linea di un'equazione, rilevata in precedenza. Nel caso, essendo le formule matematiche disgiunte all'interno di un paragrafo, il blocco sarà parte dell'equazione stessa.
- se un'equazione non viene letta separatamente dalla libreria, è necessario ciclare sull'intera stringa, in modo da isolare le formule matematiche.

Nel caso in cui la pagina contenga un'immagine possono verificarsi due scenari differenti:

1. l'immagine viene rilevata da PyMuPDF
2. la libreria non riconosce l'immagine.

Nel primo caso viene regolarmente assegnata l'etichetta alla figura. Nel secondo caso viene utilizzato un approccio alternativo: il numero di immagini e descrizioni all'interno di una pagina del documento deve risultare lo stesso. In caso contrario si esegue un algoritmo RSLA, implementato all'interno del file *image.py*.

#### 4.2. image.py

All'interno del file viene sviluppato un algoritmo alternativo per il rilevamento di figure all'interno di un documento.

```
find_image(BASE_PATH, DOC_NAME, idx,
           color_white, color_images):
```

La fase iniziale prevede di rimuovere dalla pagina PDF tutte le sequenze testuali e le figure individuate dalla libreria PyMuPDF. La rimozione avviene sovrascrivendo con un riquadro bianco varie sezioni del PDF.

```
for page in doc:
    ...
    for block in texts['blocks']:
        ...
        page2.draw_rect
        (Rect(block['bbox']),
         color=color_white, width=1.5,
         fill=color_white)
```

In questo modo, si ottiene la pagina del documento desiderata, contenente solo immagini. La pagina PDF viene trasformata in formato JPEG. Per poter lavorare sull'immagine viene eseguita una binarizzazione dei colori, utilizzando un valore di soglia arbitrario.

```
gray_painting = rgb2gray(painting)
binarized = gray_painting < 0.60
```

La presenza di alcuni "buchi" all'interno dell'immagine può tuttavia complicare la procedura di riconoscimento. Le operazioni morfologiche di *dilation* ed *erosion* consentono rispettivamente di coprire i buchi interni alla figura, mantenendo la sua forma originale.

```
square = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1]])

def multi_dil(im, num, element=square):
    ...
def multi_ero(im, num, element=square):
    ...
multi_dilated = multi_dil(binarized, 7)
area_closed = area_closing
(multi_dilated, 50000)
multi_eroded = multi_ero(area_closed, 7)
```

La fase di classificazione prevede di assegnare la stessa etichetta a gruppi di pixels adiacenti, considerati appartenenti alla stessa regione. Successivamente è possibile misurare le caratteristiche di ogni regione sfruttando un'apposita funzione; alcuni esempi sono l'area, il perimetro e la lunghezza degli assi. Queste caratteristiche risultano utili per eseguire un filtraggio sulle regioni:

quest'ultime infatti potrebbero includere del rumore.








```
label_im = label(opened)
regions = regionprops(label_im)
for num, x in enumerate(regions):
    area = x.area
    if area > 50:
        ...
```

Le regioni che presentano un'area inferiore ad un minimo stabilito risultano inutili, così come quelle il cui rapporto tra le lunghezze degli assi è sbilanciato. È possibile evidenziare le regioni di interesse generando una maschera che nasconda quelle non necessarie dell'immagine. Utilizzando i *bounding\_box* di ciascuna regione evidenziata, si individua il perimetro dell'intera immagine, che viene restituito dalla funzione.

```
minr, minc, maxr, maxc = props.bbox
...
box_image[0] = xmin
box_image[1] = ymin
box_image[2] = xmax
box_image[3] = ymax
return box_image
```

## 5. Testing

La fase di testing è stata eseguita su un dataset di 10 documenti, scaricati dalla piattaforma **SpringerLink**. Per ogni documento si è effettuato una ricerca di varie tipologie di testo all'interno del file HTML, fornendo in output un file PDF, dove sono evidenziati, tramite riquadri di colore differente, sequenze testuali ed immagini con l'etichetta della classe di appartenenza. Di seguito vengono mostrate le classi selezionate per testare il programma.

- Intestazioni 
- Descrizione Immagini 
- Descrizione Tabelle 
- Immagini 
- Formule Matematiche 
- Riferimenti bibliografici 
- Testo generico 

Al fianco di ogni classe è posto il colore che la caratterizza. I controlli riguardo la correttezza dei test vengono effettuati visivamente, valutando di volta in volta la classificazione del documento. I risultati sono riportati in tabella.

Classi	N° Test	% Correttezza
Intestazioni	83	98.79%
Descrizione Immagini	26	96.15%
Descrizione Tabelle	26	96.15%
Immagini	28	96.42%
Formule Matematiche	56	94.64%
Riferimenti Bibliografici	5	100%

Table 1. Lista delle classi selezionate, con numero di test contenuti all'interno del dataset di documenti utilizzato e percentuale di correttezza dell'algoritmo