# PDF Table Data Extraction

Claudio Pagnini
7051861
claudio.pagnini@stud.unifi.it

***Abstract* – In questo progetto si andranno ad implementare tecniche di estrazione dati da tabelle pdf tramite l'utilizzo della libreria python PyMuPDF.**
**L'articolo può esere diviso in due sezioni, nella prima si eseguono tecniche di preprocessing per estrarre le coordinate delle tabelle, nella seconda si procede all'estrazione dei dati vera e propria tramite la libreria suddetta.**

## 1 Dataset e implementazione

In questo progetto si utilizza una parte del materiale fornito da Zhong e colleghi [1]. In particolar modo è stato selezionato un dataset costituito da 140 PDF e le annotations rese disponibili in formato json. Il file delle annotazioni è un nested dictionary che ha due chiavi *primarie*: 'papers' (es. PMC3433124) e 'categories'. All'interno di ogni papers è presenta la lista delle 'pages' (es. PMC3433124_00000.pdf, PMC3433124_00006.pdf) e, per ognuna, la lista delle 'annotations'. Ciascuna annotazione ha due elementi:

- le coordinate in pixel.
- la categoria, ovvero le etichette associate ad ogni quadrupla per individuare il tipo di oggetto (es. immagine, tabella, testo)

```
PMC3433124
{'pages': ['PMC3433124_00000.pdf', 'PMC3433124_00006.pdf'],
'annotations': [[[[241, 1261, 1526, 1461], 1], [[140, 1635, 808,
1948], 1], [[857, 1577, 1524, 2048], 1], [[140, 1956, 808, 2049], 1],
[[240, 1099, 633, 1125], 1], [[241, 1150, 1526, 1235], 1], [[241,
772, 1409, 973], 1], [[240, 634, 1523, 745], 1], [[140, 437, 1203,
546], 2], [[140, 1572, 355, 1608], 2], [[240, 998, 1122, 1024], 1],
[[240, 1048, 699, 1075], 1]], [[[857, 202, 1524, 263], 1], [[140,
1244, 808, 2063], 1], [[857, 313, 1524, 531], 1], [[857, 534, 1524,
```

Fig. 1. Nested dictionary

Il progetto attraversa una doppia fase di *Pre-processing* ed una fase di *Post-processing*.

## 2 Pre-processing

Lo sviluppo del progetto prevede una prima fase di pre-processing in cui viene creato un nuovo file di annotazioni contenente le 'pages' e le coordinate relative alle tabelle: quelle con categoria 4.
Il primo step prevede di creare un dizionario con le 'pages' e tutte le coordinate relative ad esse.
Il secondo di creare un dizionario con le 'pages' e le sole coordinate relative alle tabelle. Il terzo invece semplicemente la scrittura del nuovo dizionario come nuovo documento json.

La seconda fase di pre-processing prevede l'individuazione della componente di conversione dalle coordinate in pixel alle coordinate di tipo bounding box come richiesto dalla libreria PyMuPDF, che è stata utilizzata per poter estrarre il contenuto delle tabelle. PyMuPDF permette di utilizzare il metodo *Rect* che rappresenta un rettangolo definito da quattro numeri float x0, y0, x1, y1. Si tratta dei punti relativi alle coordinate delle due diagonali, dove i primi due numeri riguardano l'angolo in "alto a sinistra", mentre i secondi due l'angolo in "basso a destra". Il controllo è stato realizzato con una precisione di $10^{-3}$ permettendo di ottenere un fattore di conversione di 0.36.

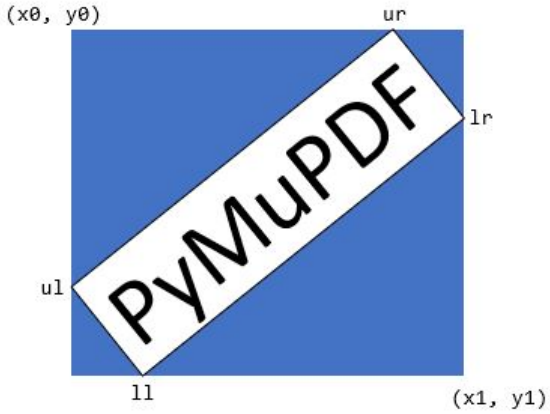



Fig. 2. BBOX Rect

### 2.1 Implentazione python

#### 2.1.1 Fase 1 - Step 1

```
mydict={}
#A new dictionary is created with 'pages' and
 ↪  related annotations
for key in d.keys():
  lst=[]
  count = 0
  for values in d[key].values(): #Values can be
   ↪  either 'pages' - .pdf - and coordinates.
    for v in values:
      if  '.pdf' in v:
        lst.append(v)
      else:
        mydict[lst[count]]=v
        count+=1
```

#### 2.1.2 Fase 1 - Step 2

```
#This method allows to find the beginning of
 ↪  coordinates required
def findBackString(string, index):
  if string[index]=='[':
    return index
  return findBackString(string, index-1)
```

```
final_dict = {}
for key in mydict.keys():
  value = mydict[key]
  listToStr = ' '.join(map(str, value))
  endTableCoord = listToStr.find(', 4]') #4 is
   ↪ the Category corrisponding to the table's
   ↪ coordinates

  if endTableCoord!=-1:
    startTableCoord = findBackString(listToStr,
     ↪ endTableCoord)
    final_dict[key]
     ↪ =listToStr[startTableCoord:endTableCoord]
```

#### 2.1.3 Fase 2

```
def ratioCalc():
  #Checks the ration between a page in pixel
   ↪ and the same page using pymupdf rect
  currentRatio = 0.0
  for path in pdfFile:
    images = convert_from_path(path)
    doc =fitz.open(path)
    page = doc[0]
    dict_ = page.get_text('dict')
    ratiow = dict_["width"] / images[0].size[0]
    if (isclose(currentRatio, ratiow,
     ↪ rel_tol=1e-3)==False):
      currentRatio = ratiow
      print("New ratio: " + str(currentRatio) +
       ↪ " for file " + path + "\n")
    ratioh = dict_["height"] /
     ↪ images[0].size[1]
    if (isclose(currentRatio, ratioh,
     ↪ rel_tol=1e-3)==False):
      currentRatio = ratioh
      print("New ratio: " + str(currentRatio) +
       ↪ " for file " + path + "\n")
```

## 3 Estrazione dati

L'estrazione dei dati avviene mediante l'utilizzo del metodo *Rect* che permette di individuare un'area specifica in cui effettuare l'estrazione dei dati e tramite l'utilizzo del metodo *get_text("words")* che permette di estrarre il testo come una lista di *parole* aventi le informmazioni relative ai bbox, come di seguito:

(x0, y0, x1, y1, "word", block_no, line_no, word_no)

### 3.1 Implementazione python

```
def extractFromPDF(path, rect):
  #Open pdf file passed via path variabile.
  #Select a rectangle thanks to the rect
   ↪ variable.
  print('\nExtracting ', os.path.basename(path)
   ↪ + '\n')
  print('------------------------------------
   ↪ --------')
  try:
    doc =fitz.open(path)
    pdfbytes = doc.convert_to_pdf()
    pdf = fitz.open("pdf", pdfbytes)
    page = pdf[0]
    words = page.get_text("words")  #Listing of
     ↪ words
    renderingText(words, rect)
  except:
    print('Error occurred')
```

## 4 Post-processing

L'output dei dati tramite il metodo *get_text("words")* non è di facile lettura, poichè contiene le coordinate x0, y0, x1, y1 ed è ordinato per x0, y1.

```
Extracting  PMC6068060_00005.pdf


----------------------------------------------

[(56.692901611328125, 314.7447204589844, 89.5323715209961, 325.66180419921875, 'patients', 0, 0, 0),
(96.19630432128906, 314.7447204589844, 105.00641632080078, 325.66180419921875, '(P', 0, 0, 1),
(111.68994140625, 314.7447204589844, 123.51841735839844, 325.66180419921875, 'for', 0, 0, 2),
(130.2215576171875, 314.7447204589844, 175.41871643066406, 325.66180419921875, 'interaction', 0, 0,
3), (177.4962921142578, 314.7447204589844, 183.2096405029297, 325.66180419921875, '=', 0, 0, 4),
```

Fig. 3. Estrazione delle word prima del Post-processing

E' stato quindi necessario effettuare un'operazione di Post-processing che permettesse di rendere il testo leggibile andando a togliere i riferimenti alle coordinate e riordinando il testo in modo che venisse ordinato in base a x0, x1.

```
Extracting  PMC6068060_00005.pdf

----------------------------------------------
Bacteria Aligner Bracket Test P
S. mutans (transformed) n Mean (SD) n Mean (SD)
Count at T0 15 9.44 (6.06) 15 12.20 (7.24) + 0.27
Count at T1 15 8.60 (6.05) 15 11.02 (7.73) + 0.35
Count at T2 15 8.87 (6.14) 15 11.09 (6.71) + 0.35
L. acidophilus (transformed) n Median (IQR) n Median (IQR)
Count at T0 15 0 (00) 15 0 (00) * 1.00
Count at T1 15 0 (00) 15 0 (00) * 0.76
Count at T2 15 0 (00) 15 0 (00) * 0.76
S. sanguinis (transformed) n Mean (SD) n Mean (SD)
Count at T0 15 23.93 (11.67) 15 32.05 (5.24) + 0.02
Count at T1 15 21.32 (10.55) 15 41.08 (10.02) + < 0.001
Count at T2 15 22.43 (9.49) 15 34.75 (7.63) + 0.001
```

Fig. 4. Estrazione delle word grazie al Post-processing

### 4.1 Implementazione python

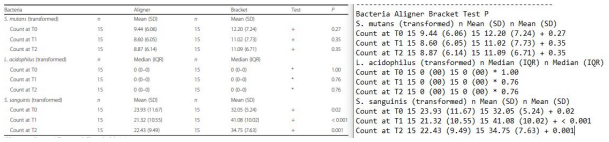
```
def renderingText(text, rect):

  """Text will be orderdered from left to
   ↪ right and from top to bottom
  """
  eeprofile = [w for w in text if
   ↪ fitz.Rect(w[:4]).intersects(rect)]
  eeprofile.sort(key=itemgetter(3, 0))
  stats = groupby(eeprofile, key =
   ↪ itemgetter(3))
  for y1, gwords in stats:
      print(" ".join(w[4]for w in gwords))
```

## 5 Implementazione python

Molte librerie python permettono l'estrazione di testo da documenti PDF. come ad esempio PyPDF2, pdfminer e PyMuPDF. PyPDF2 [2] è una libreria di Python pura capace di effettuare operazioni quali splitting, merging e cropping. Permette inoltre di estrarre il testo PDF come viene trovato dallo stream. Questo può risultare essere un problema quando il documento è complesso o, come nel nostro caso, si tratta di tabelle non strutturate. PdfMiner [3] permette di estrarre il testo non svincolato dallo stream, permette infatti di estrarre informazioni come le coordinate del testo grazie a quest'ultime. I caratteri vengono combinati per formare delle parole che, a loro volta, diventano righe e paragrafi. E' possibile manipolare la struttura dell'analisi grazie all'oggetto *LAparams()*. Il grosso svantaggio di pdfminer è che il codice risulta piuttosto complesso, soprattutto confrontato con PyPDF2 e con PyMuPDF[4]. Proprio quest'ultimo è basato su MuPDF[5], che consiste in un leggero ma consistente PDF viewer. Le potenzialità di questa libreria sono verosimili a pdfminer in quanto vengono forniti parametri diversi come "dict", "rawdict" or "xml" che permettono di ottenere output con formati differenti come, ad esempio, le coordinate. Benchè sia pdfminer che PyMuPDF siano molto simili in termini di output, è stato scelto di usare quest'ultimo per la semplicità di implementazione e per il fatto che è basato su MuPDF.

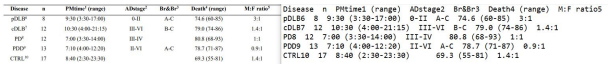
## 6 Risultati

Eseguendo l'algoritmo sui dati a disposizione è stato notato che generalmente si ottengono degli ottimi risultati. La maggior parte delle estrazioni effettuate rispecchia verosimilmente la struttura della tabella originaria.



```
Bacteria Aligner Bracket Test P
S. mutans (transformed) n Mean (SD) n Mean (SD)
Count at T0 15 9.44 (6.06) 15 12.20 (7.24) + 0.27
Count at T1 15 8.60 (6.05) 15 11.02 (7.73) + 0.35
Count at T2 15 8.87 (6.14) 15 11.09 (6.71) + 0.35
L. acidophilus (transformed) n Median (IQR) n Median (IQR)
Count at T0 15 0 (0–0) 15 0 (0–0) * 1.00
Count at T1 15 0 (0–0) 15 0 (0–0) * 0.76
Count at T2 15 0 (0–0) 15 0 (0–0) * 0.76
S. sanguinis (transformed) n Mean (SD) n Mean (SD)
Count at T0 15 23.93 (11.67) 15 32.05 (5.24) + 0.02
Count at T1 15 21.32 (10.55) 15 41.08 (10.02) + < 0.001
Count at T2 15 22.43 (9.49) 15 34.75 (7.63) + 0.001
```

Fig. 5. Comparazione con file PMC5963343_00001



```
Disease n PMtime1 (range) ADstage2 B+&B+3 Death4 (range) M:F ratio5
pDLBP 8 9:30 (3:30-17:00) 0-II A-C 74.6 (60-85) 3:1
cDLB7 12 10:30 (4:00-21:15) III-VI B-C 79.0 (74-86) 1.4:1
PD8 12 7:00 (3:30-14:00) III-IV 80.8 (68-93) 1:1
PDD9 13 7:10 (4:00-12:20) II-VI A-C 78.7 (71-87) 0.9:1
CTRL10 17 8:40 (2:30-23:30) 69.3 (55-81) 1.4:1
```

Fig. 6. Comparazione con file PMC6068060_00005

## List of Figures



## References


[1] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. "PubLayNet: largest dataset ever for document layout analysis". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. Sept. 2019, pp. 1015–1022. DOI: 10.1109/ICDAR.2019.00166.

[2] *PyPDF2*. URL: https://github.com/mstamy2/PyPDF2.

[3] *PdfMiner*. URL: https://github.com/pdfminer/pdfminer.six.

[4] *PyMuPDF*. URL: https://github.com/pymupdf/PyMuPDF.

[5] *MuPDF*. URL: https://mupdf.com/.