Trabajo de Introducción a la programación en Python

Text Mining con Facturas

Marcin Kawka, A905289
Javier Arin Reiriz A905322
Iñigo Armendia A904840

Índice de Contenidos

Índice de Códigos

Introducción

Tras la obtención del texto mediante un programa externo, se plantea la situación en la que se desea reconocer dentro de los archivos de textos procedentes del sistema de OCR diferentes siguientes datos:

- Fecha de expedición
- NIF
- Número de factura
- Base
- % IVA
- IVA CANT
- Total

Además, se han añadido a estos datos ya mencionados lo siguientes:

- Correo electrónico
- N” de Teléfono

Con el fin de facilitar al cliente el uso del programa se ha implementado una interfaz básica con la que se simplifica la ejecución del código que posibilita el trabajo. A continuación, se analizarán las diferentes secciones de código que se han utilizado para la obtención y adecuación de los datos.

Obtención de Fecha de Expedición

En esta sección se analizará el proceso de creación de código para el scraping de la **Fecha de Expedición**.

Como se puede ver en la siguiente sección de código en *Código 1*, mediante la herramienta de Regex101 se han creado las expresiones adecuadas para la detección de cada uno de los formatos correspondientes a las fechas.

```
matches = re.findall('([0-9]{1,2}-(ene|feb|mar|abr|may|jun|jul|ago|set|oct|nov|dic)-(20)?[0-9]{2,4})', \
            data[filename])

 matches = re.findall('\d{1,2}[\/|-]\d{1,2}[\/|-]\d{1,4}', \
            data[filename])

matches = re.findall('([0-9]{1,2} (de)
(enero|febrero|marzo|abril|mayo|junio|julio|agosto|septiembre|octubre|noviembre|diciembre) (de) [0-
9]{2,4})', \
            data[filename])

matches = re.findall('([0-9]{1,2}
(Enero|Febrero|Marzo|Abril|Mayo|Junio|Julio|Agosto|Septiembre|Octubre|Noviembre|Diciembre) [0-
9]{2,4})', \
            data[filename])
```

Código 1: Expresiones de Regex101 para obtención de Fechas

Para asegurar que las fechas que nos devuelve el programa son correctas, se han establecido los siguientes apartados: *Estandarización del formato de fechas, Eliminación de fechas erróneas, y eliminación de fechas erróneas.*

Estandarización del formato de fechas

Las fechas se mostrarán con el formato de Día/Mes/Año. Las funciones encargadas de hacer dichas conversiones han sido las siguientes: ***extraerFecha_1, extraerFecha_2, extraerFecha_3, extraerFecha_0,*** y se pueden observar en *Código 2.*

```
def extraerFecha_1(fech):
    a=fech.split('-')
    meses = {'ene': '01', 'feb' : '02', 'mar' : '03', "abr" : '04', 'may':'05', 'jun': '06', 'jul':'07', 'ago':'08', 'set':'09',
'oct':'10','nov':'11', 'dic':'12' }
    a[1]=meses[a[1]]
    fecha_final_aux=Logic(a)
    fecha_final='/'.join(fecha_final_aux)
    return fecha_final
```

Código 2: Estandarización del formato de fechas

Eliminación de fechas erróneas

Tras una primera detección de datos se observó cómo algunos resultados carecían de sentido debido a que excedían el número de meses o simplemente indicaban fechas futuras. Para evitar dichos resultados, se ha implementado la función ***Logic***, en el *Código 3,* que comprueba la validez de los datos obtenidos.

```
def Logic(dato): #Función para evitar resultados ilógicos.
    if float(dato[0])>31 or float(dato[1])>12 or  float(dato[2])>2025:
        vuelta=["error"]
    else:
       vuelta=dato
    return vuelta
```

Código 3: Comprobación de validez de datos

Eliminación de fechas repetidas

Para eliminar información redundante se han comprobado todas las fechas obtenidas en cada factura y se han eliminado las repetidas.

Obtención del NIF

La obtención de los NIF se ha realizado mediante la función ***findNIF*** en *Código 4* encargada de realizar un correcto scraping de estos números.

```
def findNIF(instring):
    matches = re.findall('([A-Z]-?\d{8}|\d{8}[-| ]?[A-Z]|[A-Z]-?\d{10}|\d{10}[-| ]?[A-Z])\s', \
              instring)
    return matches
```

Código 4: Obtención del NIF

Obtención de Números de Factura

Teniendo en cuenta que las facturas no siguen un patrón en concreto, se han implementado 13 códigos diferentes con Regex101 capaces de leer y guardar dichos números. En *Código 5* se puede ver el código empleado para la búsqueda de ciertos números de factura.

```
for filename in data.keys():
   intarr = []
   matches = re.findall('ES-\d{3}', \
            data[filename])
   if matches:
      intarr.append(matches)

arr.append(intarr)
newdic = dict(zip(keys,arr))
print(newdic)
```

Código 5: Expresiones Regex para los números de Factura

Para el resto de códigos se ha empleado el mismo formato, cambiando el patrón a buscar en *re.findall*. En la mayoría de los casos, se ha implementado un código que busca un patrón a partir del texto que le precede, como se puede observar en los siguientes códigos.

```
   matches1 = re.findall('(?<=FACTURA SIMPLIFICADA ).*', \
            data[filename])
```

Código 6: Número de factura de "ticket"

```
    matches2 = re.findall('(?<=Número de factura).*', \
            data[filename])
```

Código 7: Número de factura de "Prueba 1_1" y "Prueba 1_2"

```
   matches3 = re.findall('FRO\d{11}', \
            data[filename])
```

Código 8: Número de factura de "Prueba 1_5" y "Prueba 1_6"

```
    matches4 = re.findall('(?<=NÚMERO\n).*', \
            data[filename])
```

Código 9: Número de factura de "Prueba 1_9", "Prueba 1_10" y "Prueba 1_11"

```
   matches5 = re.findall('(?<=FACTURA\n).*\d{4} \d{2}', \
            data[filename])
```

Código 10: Número de factura de "Ivalíneas(2)"

```
   matches6 = re.findall('(?<=FACTURA : ).*\d{4}', \
            data[filename])
```

Código 11: Número de factura de "Prueba 1_3"

El número de factura en “Prueba 1_3”, debido a su estructura, inicialmente no se ha podido obtener correctamente el número. Por lo tanto, se ha implementado otra línea de código en la que se ha corregido dicho fallo.

```
matches7 = re.findall('(?<=\d{8}[A-Z]\n).*\d{3}\/\n\d{1} \d{3}', \
        data[filename])

matches7 = [line.replace("\n","") for line in matches7]
```

Código 12: Número de factura de “Prueba 1_3”

```
matches8 = re.findall('(?<=\d{2}\/\d{2}\/\d{4}\n).*\d{3}\/\d{4}', \
        data[filename])
```

Código 13: Número de factura de “Ivanormal(3)”

```
matches9 = re.findall('(?<=FACTURA: ).*[A-Z]\/\d{6}', \
        data[filename])
```

Código 14: Número de factura de “Ivalíneas(3)”

```
matches10 = re.findall('(?<=\d{2}\/\d{2}\/\d{2}\n).*\d{2}\/\d{3}', \
        data[filename])
```

Código 15: Número de factura de “12ocr”

```
matches11 = re.findall('(?<=Factura: ).*\d', \
        data[filename])
```

Código 16: Número de factura de “Ivalíneas(1)”

```
matches12 = re.findall('(?<=Numero: ).*\d{3}', \
        data[filename])
```

Código 17: Número de factura de “Codigoqr”

Obtención del Total

Para obtener lo que se ha pagado en total en cada factura, se ha realizado el código mostrado en *Código 18.* Se ha supuesto que, de todos los precios proporcionados en cada factura, el precio total es el mayor de dichos números, asegurando que tienen exactamente 2 decimales. Además, se han tenido en cuenta tanto notaciones Españolas como Americanas.

```
def getAllPricesAndTotal(data : str) -> tuple:
    matches = re.findall('(?:\d+?\.?)(?:\d*\,\d{2})', data)
    matches = [float(parse_decimal(d, locale="es_ES")) for d in matches]
    arr = []
    if matches:
        matches = list(filter(lambda a: a>0, matches))
        arr = arr + matches
   matches = re.findall('(?:\d+?\,?)(?:\d*\.\d{2})', data)
    matches = [float(parse_decimal(d, locale="en_US")) for d in matches]
    if matches:
        matches = list(filter(lambda a: a>0, matches))
        arr = arr + matches
    maximum = max(arr)
    return (arr, maximum)
```

Código 18: Obtención del Total de cada factura

Obtención de Base y IVA CANT

Una vez obtenido el precio total de la factura, el objetivo ha sido conseguir tanto la base como la cantidad de IVA, y para ello se han ideado dos algoritmos. En el primero de ellos, llamado twoSum, se ha supuesto que en el documento existen tanto la base como la cantidad de IVA, y por lo tanto se van a encontrar al menos un par de números que sumados den el valor total. El valor más grande pertenecerá a la base, y el más pequeño a la cantidad de IVA.

En caso de encontrar más de un par, se ha escogido aquel par cuya suma sea superior a la de otros pares, pero siempre con un valor diferente del total. Sin embargo, si el algoritmo encuentra más de una base, éste no funcionará correctamente. Por lo tanto, se ha propuesto un algoritmo alternativo de fuerza bruta, en el que a partir de todos los posibles IVAs se iteran todas las combinaciones posibles con el objetivo de encontrar la combinación exacta. Desafortunadamente, mediante este último algoritmo no se ha conseguido obtener los números necesarios para cumplir el twoSum.

El algoritmo del twoSum se muestra en el *Código 19*. Este método es envuelto por el algoritmo que nos recoge aquel resultado que tiene más probabilidad de ser el correcto. Este algoritmo se llama twoSumGetMostLikely que está situado en el mismo fichero.

```
matches = set(),
for x in arr:
    for y in arr:
        if round(x + y, 2) == target:
            if not((y,x) in matches):
                matches.add((x,y))
return list(matches)
```

Código 19: Obtención de Base y cantidad IVA

El algoritmo de fuerza bruta se muestra en el *Código 20*. Este algoritmo se encuentra dentro de la función getAllpossibleIVAConfigs y cuenta con el mismo algoritmo de selección que el anterior.

```
allPrices = allPrices + [0] * len(IVAOPTIONS)
combinationsIterator = itertools.combinations(allPrices, len(IVAOPTIONS))
IVAOPTIONSNP = np.array(IVAOPTIONS) + 1
possibleIVAConfigs = []
for combination in combinationsIterator:
   if np.dot(IVAOPTIONSNP, np.array(combination)) == total:
      possibleIVAConfigs.append(combination)

if not possibleIVAConfigs:
   return possibleIVAConfigs
max = -1
idx = 0
for i in range(len(possibleIVAConfigs)):
   for el in possibleIVAConfigs[i]:
      (max,idx) = (el,i) if (el > max and el != total) else (max,idx)
return possibleIVAConfigs[i]
```

Código 20: Algoritmo alternativo del código 19

Obtención de %IVA

Una vez hallada la cantidad del IVA y de la base, se ha calculado el IVA mediante una simple operación matemática.

Obtención Números de teléfono

Tras obtener todos los datos pedidos en el enunciado del trabajo, para proporcionar una información más completa al cliente, se han obtenido tanto los números de teléfono, como correos electrónicos correspondientes a cada factura.

```
   matches13 = re.findall('9\d \d{3} \d{2} \d{2}\s| 9\d{2} \d{6}\s| 9\d{2} \d{3} \d{3}\s| 9\d{8}\s| 9\d{2} \d{2}
\d{2} \d{2}\s', \
            data[filename])

   matches13 = [line.strip().strip('\n') for line in matches13]
```

Código 21: Números de teléfono

Obtención de Correos electrónicos

```
   matches14 = re.findall('(?:\w+\.)?\w+@\w+.\w+', \
            data[filename])
```

Código 22: Correos Electrónicos

Conclusiones

Se ha logrado crear un programa funcional que es capaz de extraer una gran variedad de datos, los cuales a su vez tienen diferentes formatos. Además, mediante la implementación de una interfaz gráfica se facilita el uso del programa.

El punto negativo que tiene el programa es su sensibilidad al cambio de formato en los datos que se desean extraer. Esto se debe a que todas las expresiones de Regex que se han introducido en el código son específicas para una serie de formatos determinados, por lo que un cambio en la forma de los datos haría que el sistema no los detectase. Aun así, este problema es fácilmente solventable gracias a la facilidad con la cual se pueden incluir nuevas expresiones al código de detección.

Instrucciones de uso

Se adjunta el documento LEEME.txt en el que se explican los pasos a seguir para el correcto funcionamiento del programa. Además, se adjunta una carpeta que contiene capturas de pantalla para facilitar su uso.