Extraer desde una fuente información para minería de datos

A parte de la información que ya nos venga preparada en ficheros como vimos en el apartado anterior. Podemos encontrarnos con la necesidad de extraer esta información de otro tipo de fuentes como:

- bases de datos
- páginas web
- ficheros PDF
- texto
- ... Vamos a ver algunos ejemplos.

Extracción de datos de Sqlite

Para mostrar un ejemplo de como acceder a una base de datos SQL usaremos SQlite por su simplicidad. Salvando la diferencia de la autentificación para poder acceder a otras bases de datos, el proceso es similar para todas.

Por ejemplo, sobre la base de datos provista como ejemplo en este tutorial (https://www.sqlitetutorial.net/sqlite-sample-database/), vamos a sacar una tabla con las pistas de audio (*tracks*) de la lista de reproducción (*playlist*) denominada 'Classical' que duren menos de 2 minutos.

Si no tienes instalado el interfaz de Sqlite para Python puedes instalarlo con:

In [ ]:
```
#!pip install pysqlite3 # quitar el comentario (# inicial) si se q
uiere intentar instalar desde dentro de Jupyter
```

In [ ]:
```
import pandas as pd
import sqlite3
conexion = sqlite3.connect('chinook.db')
cursor = conexion.cursor()
consulta = """
    SELECT playlists.Name, tracks.AlbumId, tracks.Name, tracks.Com
poser, tracks.MediaTypeId,
           tracks.Milliseconds, tracks.Bytes, tracks.UnitPrice
        FROM playlists, playlist_track, tracks
        WHERE playlists.name == 'Classical' and
              playlists.PlaylistId == playlist_track.PlaylistId an
d
              playlist_track.TrackId == tracks.TrackId and
              tracks.Milliseconds < 120000
"""
tabla = pd.read_sql_query(consulta, conexion)
tabla
```

Si quisiesemos saber la densidad de información de los archivos de música (KB/s), podríamos hacerlo de dos formas:

- cambiando la consulta SQL, o
- calculándolo con Pandas sobre la tabla preparada.

Ante la decisión de como implementarlo, merece la pena pararse a pensar qué partes del proceso delegamos en el sistema de la base de datos y qué partes es más eficiente hacer en código. En general, al descubrir una nueva herramienta o técnología, muchas veces nos empeñamos en intentar exprimir al máximo esta herramienta cuando sería más fácil hacerlo con otra.

Una regla que suele funcionar bien, es dejar para la base de datos la función para la que está optimizada (hacer operaciones join, extraer y filtrar datos) y resolver la lógica de negocio (cálculos y procesos más avanzados) con herramientas de programación. No obstante, la decisión dependerá del caso concreto, teniendo en cuenta diversos factores, como los volúmenes de datos que deban moverse, el tipo de cálculos, la base de datos que estamos usando... Por otra parte, el factor más importante será el número de veces que se va a repetir la operación. En el contexto de la ciencia de datos, muchas veces los datos se van a extraer una única vez. En esos casos, la decisión debe basarse en lo que nos resulte más fácil de programar. Normalmente no merecerá la pena ponerse a optimizar. Por supuesto, siempre habrá excepciones, por ejemplo, si trabajamos con volumenes de datos muy grandes o si la optimización es muy sencilla.

Ejercicio: Calcular la densidad de la información (lo que ocupa cada segundo de sonido) en kB/s para las canciones seleccionadas de ambas formas.

In [ ]:
```
# Introduce aquí el código que, usando Pandas, le añade a la tabla una columna con la densidad de información
```

In [ ]:
```
# Introduce aquí el código con una nueva consulta SQL que lo calcule directamente en una columna nueva de la tabla
```

Solución: las densidades salen todas entorno a 17, 37 o 43kB/s. Ni que decir tiene que de las dos formas debe salir el mismo resultado.

Webscrapping: texto y PDF

Texto de un HTML

In [ ]:
```
import urllib.request
with urllib.request.urlopen('http://ax5.com/spanish/2020/10/01/sistema-de-doble-sobre-voto-por-correo.html') as flujo_web:
    html = flujo_web.read().decode('utf-8')
print(html)
```

En la librería estándar de Python tenemos el módulo html (https://docs.python.org/3/library/html.html) que nos permite procesar con todo detalle un documento html. Para trabajos donde la estructura y los *tags* (marcas html) son importantes, puede sernos muy útil. Sin embargo, al cientifico de datos pueden serle muy útiles otras soluciones más sencillas.

Las expresiones regulares (https://docs.python.org/3/library/re.html) nos dan una gran potencia para procesados de texto simples y rápidos. Si no se ha trabajado con ellas anteriormente, recomendamos la lectura del tutorial: Regular Expression HOWTO (https://docs.python.org/3/howto/regex.html).

Con ellas podemos, por ejemplo, extraer solo el texto del html anterior:

In [ ]:
```
import re
solo_texto = re.sub("<[^>]*>", "", html) # elimina las marcas html (tags)
print(solo_texto)
```

PDF

Como con muchas otras cosas, Python tiene buenas herramientas para trabajar a fondo con archivos PDF. Puede ser interesante leer este artículo que revisa diferentes herramientas para extraer texto de archivos PDF (https://towardsdatascience.com/pdf-preprocessing-with-python-19829752af9f).

No obstante, muchas veces nos sirve con una solución más sencilla y rápida. En el siguiente ejemplo real vemos como extraer fácilmente el texto de un archivo PDF descargado de la web usando herramientas del sistema operativo. Concretamente usamos las herramientas de software libre (https://www.gnu.org/philosophy/free-sw.es.html) `wget` (https://www.gnu.org/software/wget/) y `pdftotext` (http://www.xpdfreader.com/pdftotext-man.html).

Si no las tienes instaladas, en los sistemas basados en Debian pueden instalarse con:

In [ ]:
```
#!sudo apt install wget poppler-utils # quitar el comentario (# inicial) si se quiere intentar instalar desde dentro de Jupyter
```

Extracto de un código que extrae información de las guías docentes de la Universidad de Córdoba.

Para simplificar solo extraeremos información de una guía dando valores a variables que proceden de bucles que hacen un recorrido por todas las guías.

In [ ]:
```
import os

# Constante de configuración (por eso está en mayúculas)
CURSO = '2021-22'

# Variables que en el código original son parámetros de la función
que procesa el texto
code = '101332'
filename = code + 'es_' + CURSO
url_guia = 'http://www.uco.es/eguiado/guias/' + CURSO + '/' + file
name + '.pdf'

# Descarga y extracción del texto de la guía en PDF
os.system('wget ' + url_guia)
if os.path.isfile(filename + '.pdf'):
    os.system('pdftotext ' + filename + '.pdf')

    with open(filename + '.txt', 'r') as file:
        line = file.readline()
        while line:
            # Aqui se procesa el texto para extraer los datos que
nos interesan. Esta es otra forma
            #  de leer el texto. En vez de cargarlo entero en memo
ría lo vamos procesando línea a
            #  línea. Puede ser más eficiente si los ficheros son
muy grandes.
            # (en el ejemplo vamos simplemente a imprimirlo)
            print(line)
            line = file.readline()
```

Crear una bolsa de palabras (*Bag of words*)

El modelo de la bolsa de palabras (*Bag of words*) es una representación de un texto por un vector que indica el número de veces que aparecen ciertas palabras en el texto. Veamos un ejemplo de como crearlos rápidamente en Python, usando solamente la librería estándar.

In [ ]:
```
# Quitamos los espacios iniciales y finales porque re.split() no l
os elimina
#  nota: str.strip() no sirve porque dejaría algunos caracteres no
deseados
solo_texto = re.sub('^\W+|\W+$', '', solo_texto)

# Consideramos separadores todos los caracteres que no puedan ser
palabras o números
lista_palabras = re.split('\W+', solo_texto)
lista_palabras[:10]
```

Ya tenemos una lista con todas las palabras del texto limpias. Vamos a contar cuantas veces aparece cada una con la estructura de datos `collections.Counter` (https://docs.python.org/3/library/collections.html?highlight=counter#collections.Counter) que ya conoceis de la asignatura anterior.

In [ ]:
```
import collections
bag_of_words_diccionario = collections.Counter(lista_palabras)
bag_of_words_diccionario
```

Podemos así imprimir por ejemplo las palabras más frecuentes.

In [ ]:
```
for palabra, apariciones in bag_of_words_diccionario.items():
    if apariciones > 5:
        print('{:<9} {:>2}'.format(palabra, apariciones))
```

Con estos datos, si eliminamos las *stop-words*, que son las palabras más frecuentes de un idioma y no aportan el significado principal ('de', 'para', 'por', 'la', 'se', 'una', 'un', ...), podemos intuir el tema del documento. Resulta sorprendente que con una idea tan sencilla se pueda extraer información tan útil.

Para procesado más avanzado de text es interesante conocer el Natural Language Toolkit (https://www.nltk.org/).

Ejercicio: Experimenta descargando y extrayendo información que consideres interesante de algún documento publicado en internet (en PDF o en HTML)