ENLACES RECOMENDADOS

JSoup. https://jsoup.org/

OpenNLP. https://opennlp.apache.org/

GUÍA 2: EXTRACCION AUTOMATICA DE INFORMACION DE GOOGLE PLAY

1. INTRODUCCIÓN

El volumen de apps plublicadas en los mercados en línea representan una gran posibilidad para análisis de datos en lo correspondiente a funcionalidades, rating, comentarios de usuarios, etc. En particular, el análisis de mercado es una actividad obligatoria durante el proceso de concepción de una app, y obviamente durante evolución y mantenimiento, para identificar las características que pueden ser diferenciadoras, o las funcionalidades que se deben tener en una app para recibir buenas comentarios y ratings.

Sin embargo, el volumen de datos disponible es a su vez un inconveniente para el análisis, dado que la extracción manual de datos de los mercados es costosa en términos de tiempo. Para esto, se han creado servicios de analítica web que contantemente y automáticamente extraen la información de las páginas web de los mercados y la ofrecen (obviamente a un costo) consolidada o con funcionalidades de búsqueda y navegación a los interesados, como en el caso de 42 matters (https://42matters.com/api) y app annie (https://www.appannie.com/en/). Otros portales como statista (https://www.statista.com/topics/1002/mobile-app-usage/) se orientan a la construcción de reportes gratuitos pero de corte general.

En esta guía usted aprenderá las bases para extraer información automáticamente de un mercado de apps móviles (Google Play). Los mismos conceptos y técnicas aplican para cualquier otro mercado que publique información en formato HTML y vía el protocolo HTTP.

2. EXTRAER LA LISTA DE APLICACIONES TOP EN UNA CATEGORÍA

Google play publica una lista de aplicaciones recomendadas, conocida como la Editor´s choice list (https://play.google.com/store/apps/topic?id=editors_choice), que contiene tanto aplicaciones gratuitas como pagas. La Editor´s choice lista puede ser usada como referencia a la hora de identificar las apps que son tendencia y las funcionalidades que están siendo valoradas bien tanto por Google como por los usuarios. Por otro lado, Google Play, permite visualizar la lista de las aplicaciones con mejor rating (pagas o gratuitas independientemente) organizadas por categoría. Por ejemplo, las aplicaciones top, pagas, de la categoría FINANCE, se pueden consultar en la URL https://play.google.com/store/apps/category/FINANCE/collection/topselling_paid, y las top, gratuitas, de la misma categoría, se pueden consultar en la URL https://play.google.com/store/apps/category/FINANCE/collection/topselling_free.

Dado que esta información está publicada en formato HTML y los desarrolladores/diseñadores del sitio usan markups, clases, & ids en los sitios para organizar el contenido, se puede hacer uso de herramientas conocidas como crawlers/scrapers[1] que permiten programáticamente cargar el contenido HTML en una URL y explorarlo a través del uso de selectores. En esta guía usted va a usar la librería jsoup (https://jsoup.org/) que permite implementar los dos comportamientos mencionados anteriormente (crawler/scraper). Para esto, primero diríjase a la página de jsoup, descargue el jar, y cree un proyecto java que tenga a la librería como dependencia.

Jsoup permite cargar un documento HTML tanto local como remoto haciendo uso de una URL. Suponga que estamos interesados en extraer información de la lista aplicaciones top, pagas, de la categoría FINANCE; la llamada a la API para crear el respectivo documento es la siguiente:

```
Document doc = Jsoup.connect("https://play.google.com/store/apps/category/FINANCE/collection/topselling_paid").
        timeout(0).get();
```

Figura 1. APIs para cargar un documento HTML con JSoup

[1] Un crawler explora una página web a través de los hiperenlaces, y un scraper extrae información de una página web usando selectores o tags (e.g., todo el contenido de texto en los tags <p>)

Note que la llamada estática al método connect, tiene como argumento a la URL deseada, y se acompaña de una cadena de llamadas a los métodos timeout y get; la llamada a timeout es necesaria para evitar problemas de latencia de red o fallos de acceso a la URL cuando el timeout ha sido alcanzado; por eso se sugiere usar un valor timeout de 0 que indica a JSoup que no hay timeout. Corra la línea de código anterior e imprima el objeto doc en la pantalla, usted verá todo el contenido HTML de la página de aplicaciones top, pagas, de la categoría FINANCE (Figura 2).

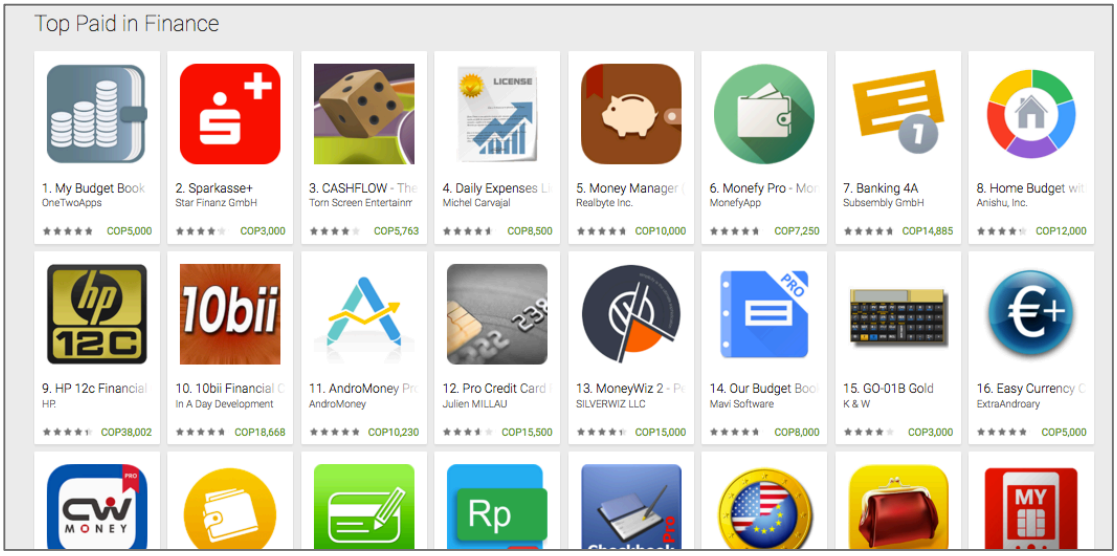

Figura 2. Ejemplo de lista de aplicaciones top, pagas, de la categoría FINANCE en Google Play

Un documento en Jsoup es una colección de "Elements" que a su vez pueden tener más "Elements" siguiendo la jerarquía de inclusión del HTML analizado. En este caso estamos interesados en extraer la información de las apps, pero la lista de aplicaciones top solo proporciona el logo, nombre, rating promedio, y costo. Para poder acceder al detalle de cada app, se necesita simular un click en el logo de las apps, o inferir la URL de acceso a cada app. En Google Play, las URLs de acceso a los detalles de cada app siguen el formato https://play.google.com/store/apps/details?id=<id_de_la_app>[2]. Si usted revisa el código fuente de la lista en la Figura 2, se dará cuenta que los ids de las apps aparecen ahí y en particular la URL relativa para acceder al detalle de cada app (Figura 3) se encuentra en un enlace con clase card-click-target. Verifique que esa clase solo se usa para enlaces que llevan al detalle de las apps.

```
style="display:none">1</div> <button class="price buy id-track-click id-track-impression" data-server-
cookie="CAIaKgomEiQKHm1pYy5hcHAuZ2FzdG9zZGlhcmlvc19saWNlbmNpYRABGANCAA==" data-uitype="200">    <span class="display-
price">COP8,500</span>   </button>    </span>     </span> </div>  <div class="description">  This is the license of the
application Daily Expenses 2 and 3 <span class="paragraph-end"></span>  <a class="card-click-target"
href="/store/apps/details?id=mic.app.gastosdiarios_licencia" aria-hidden="true" tabindex="-1"> </a>   </div> </div>
```

Figura 3. Código fuente de lista de aplicaciones

[2] El id de cada apps es el mismo package name usado por el desarrollador

Dado que hay una clase que solo se usa con los enlaces de los detalles, de las apps, podemos recorrer el documento JSoup de la página y solicitar el atributo href para los tags con la clase "card-click-target"; note que hay múltiples tags de enlace por app, pero con la misma URL, entonces se deben guardar los resultados en un conjunto en vez de una lista. JSoup tiene varios métodos de consulta a un documento, incluyendo el método para seleccionar elementos de acuerdo con una clase Document. getElementsByClass(…); adicionalmente a cada elemento (e.g., un tag div) se le pueden consultar los atributos usando el método Element.attr(…). En nuestro caso necesitamos los elementos con clase "card-click-target" y de cada uno necesitamos el atributo href. El siguiente snippet de código hace la tarea y guarda las URLs de cada app en un hashset (para evitar duplicados). Note que el atributo href tiene URLs relativas, por eso se les concatena el dominio de Google Play:

```
Document doc = Jsoup.connect("https://play.google.com/store/apps/category/FINANCE
        timeout(0).get();

HashSet<String> hrefs = new HashSet<String>();

Elements anchors = doc.getElementsByClass("card-click-target");
for (Element element : anchors) {
    hrefs.add( "https://play.google.com/" + element.attr("href").toString() );
}
```

Figura 4. Ejemplo de código para extraer elementos por clase, y luego obtener un atributo de cada elemento

3. EXTRAER DETALLES DE CADA APP EN LA LISTA

La Figura 4 muestra el fragmento de código que permite obtener la URL de las apps en la lista deseada. Con esas URLs, de forma iterativa se pueden cargar documentos JSoup, y a su vez extraer información de forma similar a cómo se extrajeron las URLs en el paso anterior. Para entender el proceso, use como referencia la página de detalle de la aplicación DaviPlata[3]. La información de interés en este caso es: (i) descripción de la app, (ii) cambios recientes (what´s new), (iii) rating promedio, (iv) comentarios, (v) número de ratings por estrellas. Revise el código fuente de la página y busque los selectores y tags asociados con la información de interés. La siguiente tabla muestra los selectores:

Información	**Selector**
Nombre	class="id-app-title"
Número de ratings	class="rating-count"
Rating promedio	class="score"
Descripción	itemprop="description"
Cambios recientes	class="recent-change"[4]
Ratings con 5 estrellas	class="rating-bar-container five"
Ratings con 4 estrellas	class="rating-bar-container four"

[3] https://play.google.com/store/apps/details?id=com.davivienda.daviplataapp
[4] En este caso, se tienen varios elementos por app con ese selector

Para efectos de guía, se presentará el código de ejemplo para extraer la información de la descripción. En este caso, el selector no es la clase, sino el atributo itemprop. JSoup tiene un método general para localizar elementos usando selectores: Document.select(…). Para buscar por atributo con valor la sintaxis del selector es [attr=value], por lo tanto en nuestro caso de ejemplo el selector a usar es [itemprop='description']. Mas información acerca de los selectores usados por JSoup se encuentra aquí: https://jsoup.org/cookbook/extracting-data/selector-syntax

El siguiente fragmento de código muestra cómo extraer la descripción de cada app en la lista deseada. Note que el doble ciclo es para evitar URLs de detalle duplicadas; por eso las URLs extraídas de la lista se guardan en un conjunto y luego se recorre el conjunto. Para evitar el doble ciclo se podría usar un condicional en el ciclo inicial.

```
Document doc = Jsoup.connect("https://play.google.com/store/apps/category/
        timeout(0).get();

Document detailDoc = null;

HashSet<String> hrefs = new HashSet<String>();
String href = null;

Elements anchors = doc.getElementsByClass("card-click-target");

String description = null;
for (Element element : anchors) {
    href = "https://play.google.com/" + element.attr("href").toString() ;
    hrefs.add( href);
}


for(String url: hrefs){
    detailDoc = Jsoup.connect(url).timeout(0).get();
    description = detailDoc.select("[itemprop='description']").text();
}
```

Figura 5. Ejemplo de código para extraer elementos usando un selector

4. ANALISIS DE TEXTO

Los pasos anteriores ilustran cómo extraer información automáticamente de páginas web haciendo uso de la librería JSoup. En el caso de mercados de apps, hay limitaciones como el número de comentarios que se pueden ver; sin embargo, la información en la descripción, los ratings, y los nuevos cambios pueden usarse para identificar las funcionalidades de las apps que tienen mejor rating, o las funcionalidades de las apps con bajo rating. Para esto se puede hacer uso de herramientas para análisis de texto que permitan identificar sentencias, palabras frecuentes, n_gramas frecuentes. Una de estás

herramientas es Apache OpenNLP[5] (https://opennlp.apache.org) que más allá de simples tareas de selección de sentencias y tokenización, permite también hacer categorización de texto. Dado que estos temas están fuera del alcance del curso, se sugiere leer el tutorial de OpenNLP que se encuentra en

https://opennlp.apache.org/documentation/1.7.1/manual/opennlp.html

5. RESUMEN

Las herramientas aquí presentadas son suficientes para empezar a diseñar herramientas automáticas que soporten la concepción de una app, desde el punto de vista de análisis de mercado. En esa medida, su equipo de desarrollo debe implementar herramientas que le permitan extraer información de los mercados dominantes (Google Play y App Store) directamente o a través de repositorios como app annie, y realizar analítica sobre esa información.

Bono: Se darán puntos adicionales a los equipos que presenten analítica basada en análisis de texto.

[5] Hay herramientas más avanzadas como la suite de herramientas para procesamiento de lenguaje natural de Stanford: http://nlp.stanford.edu/software/

VERSIONES DEL DOCUMENTO

VERSIÓN	AUTOR(ES)	FECHA
1.0	MARIO LINARES VASQUEZ	LUNES 30 ENERO, 2017
2.0	MARIO LINARES VASQUEZ	DOMINGO 13 AGOSTO, 2017