
Extracción de Datos Estructurados a partir de páginas web (web scrapping)

Scrappers

- Mucha de la información en la web (p.ej. listas de productos o servicios) está representada en estructuras, porque las páginas son plantillas que se rellenan con datos.
 - Esta estructura regular se puede aprovechar para extraer los datos nuevamente como registros.
 - Aplicaciones: p.ej. Compra comparativa, meta buscadores.
 - Un scrapper es un programa que extrae datos de páginas web.
 - Vamos a hablar de:
 - Scrappers basados en reglas
 - Scrappers inducidos (aprendizaje supervisado)
 - Scrapping Automático (Aprendizaje semi supervisado)
-

Dos tipos de estructuras de página

- **Páginas de listados**

- Cada página contiene uno mas listas de registros con datos
 - Cada lista se corresponde a una región específica en la página, y los registro con datos pueden ser chatos o anidados.

- **Páginas de Detalles**

- Cada página tiene los datos de un sólo objeto...
- ...pero además contiene un montón de información que no tiene relación directa con el objeto.

Extracción de datos



imagen1	Cabinet Organizers by Copco	9-in	Round Turntable: White	*****	\$4.95
imagen1	Cabinet Organizers by Copco	12-in	Round Turntable: White	*****	\$7.95
imagen2	Cabinet Organizers	14.75x9	Cabinet Organizers (Non-Skid): White	*****	\$7.95
imagen3	Cabinet Organizers	22x6	Cookware Lid Rack	****	\$19.95

Modelo de datos subyacente

- La mayoría de las páginas web pueden modelarse como compuestas de **relaciones anidadas**, donde cada item tiene un tipo y una serie de valores asociados (que puede ser compuestos).
 - *name* (of type *string*),
 - *image* (of type *image-file*), and
 - *differentSizes* (a *set* type), consists of a set of tuples with the attributes:
 - *size* (of type *string*), and
 - *price* (of type *string*).

```
tuple product ( name:      string;
                 image:    image-file;
                 differentSizes: set ( size:  string;
                                     price:  string; ))
```

HTML versus estructura de los datos

- HTML no tiene tags para cada tipo de datos, porque no fue creado para representar estructuras de datos, sino de páginas. Casi cualquier tag de HTML tag puede corresponderse a cualquier tipo de datos.
 - Sin embargo los items en registros son representados de manera regular en la página, para distinguirlos de otros items y resaltar lo importante.
 - Un item a su vez puede estar particionado en varios sub- grupos disjuntos de items. Cada sub-grupo se corresponde con un grupo de atributos y usualmente tiene una codificacion distinta en HTML.
-

HTML versus estructura de los datos

- Este esquema no cubre todos las representaciones posibles de datos en HTML, pero sí a la mayoría.
- Además, los datos en una página web pueden no solo diferenciarse por el encoding en HTML,, sino también por palabras y signos de puntuación.
- **Ejemplo:**

Restaurant Name: **Good Noodles**

- 205 Willow, *Glen*, Phone 1-773-366-1987
- 25 Oak, *Forest*, Phone (800) 234-7903
- 324 Halsted St., *Chicago*, Phone 1-800-996-5023
- 700 Lake St., *Oak Park*, Phone: (708) 798-0008

Extracción por reglas

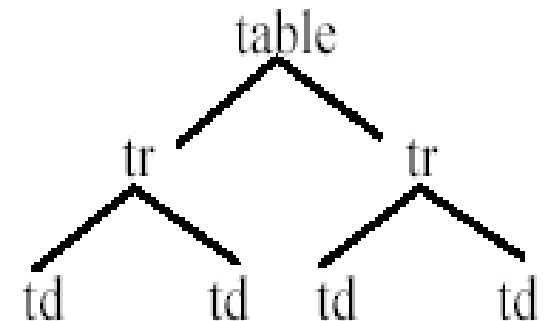
- Como una página web es texto, la forma más simple de extracción es usando expresiones regulares.
 - Las expresiones regulares pueden combinarse para crear plantillas anidadas .
 - **Extracción:**
 - Dada una expresión regular, es posible crear automáticamente un micro-programa (llamado autómatas finitos) que de manera eficaz es capaz de reconocer secuencias de texto que se correspondan a la expresión regular dentro de la página web.
 - La expresión regular puede contener información sobre el contexto que rodea a los datos, y áreas que se corresponden a los datos a extraer, llamadas **grupos de captura**.
 - P.ej. La expresión regular
 - `[]+\((([0-9]+)\)[]*-(+[0-9]{3}-[0-9]{4})`
 - Detecta los número de teléfono con la forma **(3_dígitos_de_característica)-3_dígitos-4_dígitos** y extrae **2 grupos de captura: la característica y el número**.
-

HTML como un árbol: el DOM

- La mayoría de los tags de HTML funcionan en grupo (uno de apertura y otro de cierre, p.ej. `<a>` y ``). Entre un tag de apertura y otro de cierre puede haber otros pares de tags, formando así una estructura de árbol:
- Cada par de tags de apertura y cierre es un nodo del DOM (Document Object Model), y los tags entre medio son nodos hijos de ese nodo.

Ejemplo de HTML: equivalencias

1	<table>	left	right	top	bottom
2	<tr>	100	300	200	400
3	<td>...</td>	100	300	200	300
4	<td>...</td>	100	200	200	300
5	</tr>	200	300	200	300
6	<tr>				
7	<td>...</td>	100	300	300	400
8	<td>...</td>	100	200	300	400
9	</tr>	200	300	300	400
10	</table>				



Construyendo el árbol de HTML

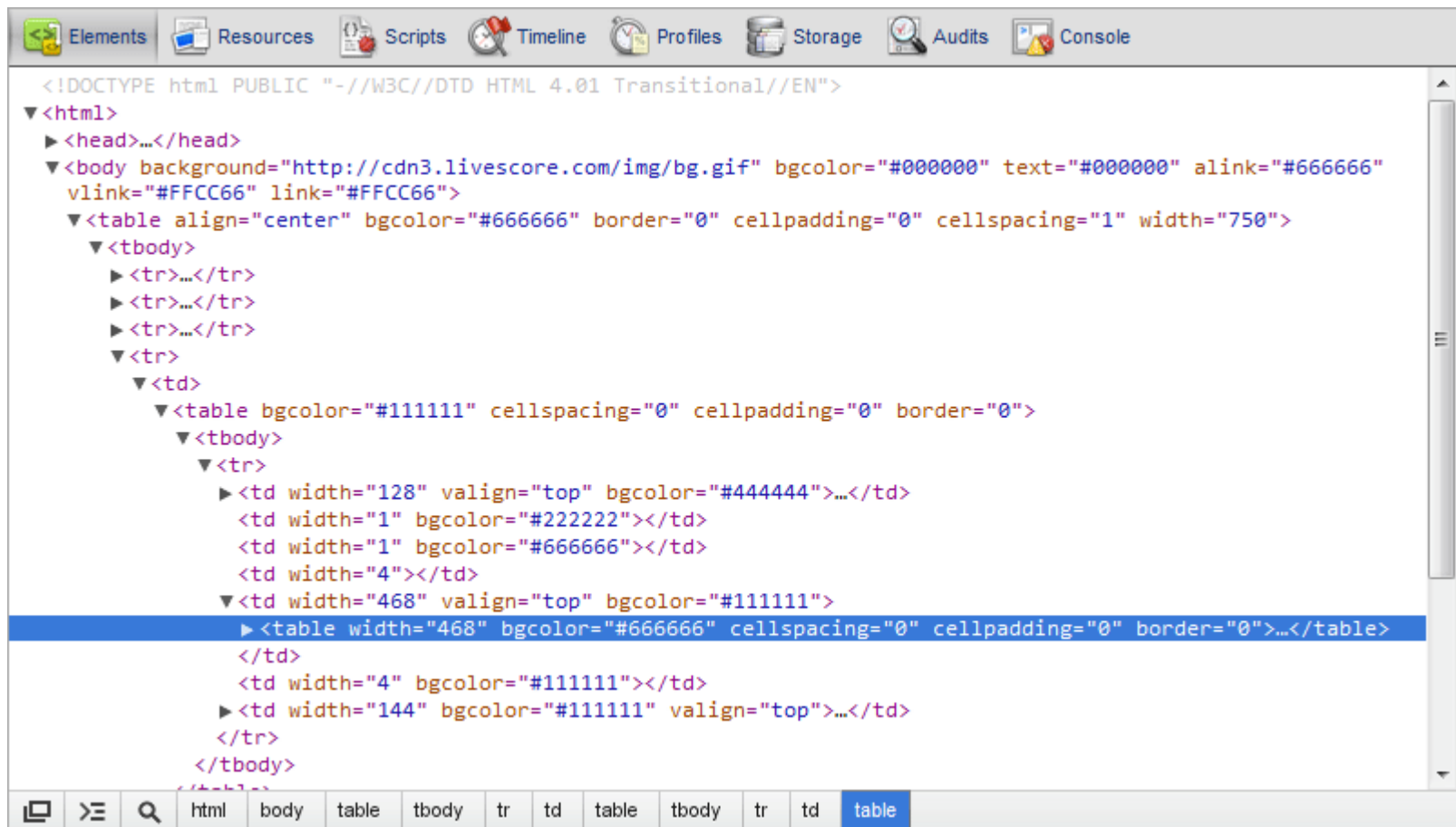
- Si el HTML está bien formado, construir el árbol de DOM es trivial. Lamentablemente, no siempre es así.
- Los browsers son resistentes a HTML mal formado. Muchas páginas web están mal formadas y requieren “limpieza”:
- Algunos tags no requieren tags de cierre obligatoriamente (p.ej. `<hr>`, `<p>`) aunque los tags de cierre existen.
- Las páginas muchas veces no contienen los tags de cierre aunque sean obligatorios.

Construyendo el árbol de HTML

- Corregir errores en HTML no es simple, pero si el browser puede mostrar la página visualmente, eso significa que es posible construir un árbol de DOM. La posición de estos tags de cierre se puede inferir para “limpiar” el HTML. Un programa muy popular para limpiar HTML es Tidy (<https://www.html-tidy.org/>).
- Esto se puede complicar aún más, porque el contenido de la página puede ser generado dinámicamente usando código javascript que se ejecuta luego de cargada la página.

XPaths

- Xpaths son expresiones sobre la estructura del árbol de HTML (asumiendo que código HTML está bien formado).
- Las expresiones XPaths son fáciles de encontrar usando las herramientas de desarrollo dentro de un browser, pero no resisten cambios en la estructura de la página.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>...</head>
<body background="http://cdn3.livescore.com/img/bg.gif" bgcolor="#000000" text="#000000" alink="#666666" vlink="#FFCC66" link="#FFCC66">
  <table align="center" bgcolor="#666666" border="0" cellpadding="0" cellspacing="1" width="750">
    <tbody>
      <tr>...</tr>
      <tr>...</tr>
      <tr>...</tr>
      <tr>
        <td>
          <table bgcolor="#111111" cellspacing="0" cellpadding="0" border="0">
            <tbody>
              <tr>
                <td width="128" valign="top" bgcolor="#444444">...</td>
                <td width="1" bgcolor="#222222"></td>
                <td width="1" bgcolor="#666666"></td>
                <td width="4"></td>
                <td width="468" valign="top" bgcolor="#111111">
                  <table width="468" bgcolor="#666666" cellspacing="0" cellpadding="0" border="0">...</table>
                </td>
                <td width="4" bgcolor="#111111"></td>
                <td width="144" bgcolor="#111111" valign="top">...</td>
              </tr>
            </tbody>
          </table>
        </td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

...lo que el browser "ve" está limpiado,
no es lo que contiene el HTML original

■ Xpath: `//table[1]//tr[4]//table//tr[1]/td[5]//table//tr`

Inducción de scrappers

- Inducir scrappers es usar machine learning para generar automáticamente reglas de extracción
 - El usuario marca visualmente a los items de interés en ejemplos de entrenamiento.
 - El sistema aprende reglas de extracción a partir de esos ejemplos.
 - Las reglas se aplican para extraer items de otras páginas.
- Existen varios métodos de inducción de scrappers, nosotros vamos a ver uno de ejemplo.

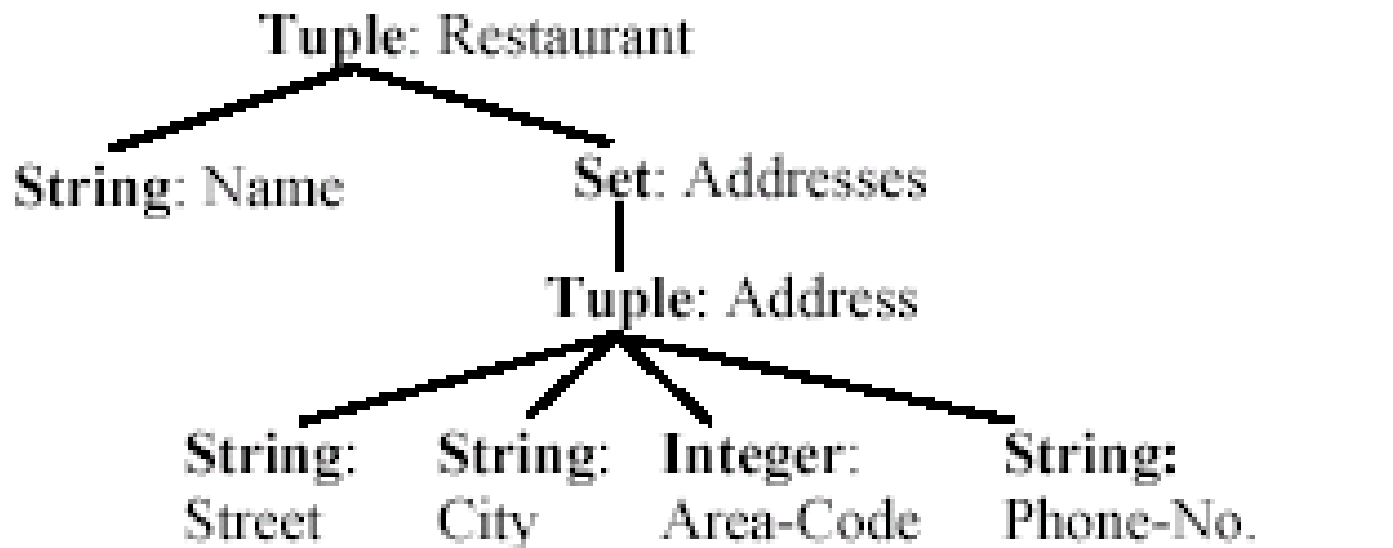
Stalker: Un sistema de inducción jerárquica de scrappers

- Aprendizaje jerárquico de scrapping
 - Extrae items agrupados en jerarquías
 - Es útil para estructuras anidadas de registros.
- Asume que cada ítem que se extrae es independiente de otros items en el mismo nivel.
- Cada item es extraído usando 2 reglas:
 - Una regla de comienzo (**start rule**) para detectar el comienzo del item a extraer.
 - Una regla de fin (**end rule**) para detectar donde termina el item a extraer.
 - Para un item que contiene varios items (una lista), una regla de iteración es necesaria para separar los items contenidos.

Representación jerárquica: árbol de tipos

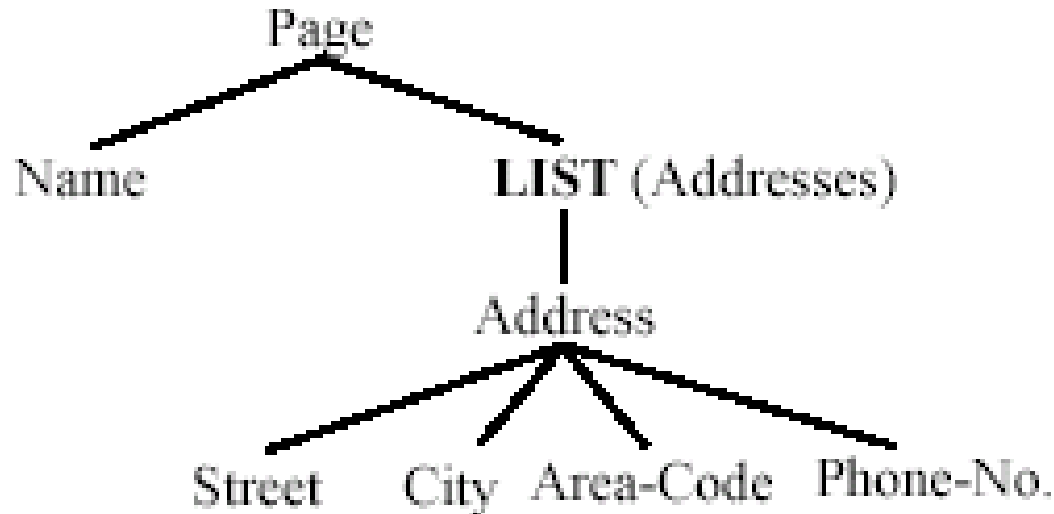
Restaurant Name: **Good Noodles**

- 205 Willow, *Glen*, Phone 1-773-366-1987
- 25 Oak, *Forest*, Phone (800) 234-7903
- 324 Halsted St., *Chicago*, Phone 1-800-996-5023
- 700 Lake St., *Oak Park*, Phone: (708) 798-0008



Extracción de datos

- La extracción funciona sobre un esquema del contenido de la página, definido como una estructura de árbol.
- Cada tipo del esquema se basa en los nodos de más arriba.



- Para extraer cada item (un nodo del esquema), el scrapper necesita una regla que lo extraiga del nodo padre.
- Un nodo del esquema se mapea a uno o más nodos de HTML.

Landmarks

- Las reglas de extracción están basadas en la idea de **landmarks**.
 - Un landmark es una **secuencia consecutiva** de tokens.
- Los landmarks se usan para localizar el comienzo y final de un item.
- Las reglas operan sobre landmarks y sobre el resultado de otras reglas.
- Una regla puede contener landmarks y operadores. Los **operadores** sirven para saltar tags, buscar un tag en particular, modificar texto, etc.

Un ejemplo

- El texto de ejemplo, en HTML, se ve así:

```
1: <p>Restaurant Name: <b>Good Noodles</b><br><br><ul>
2: <li>205 Willow, <i>Glen</i>, Phone 1-<i>773</i>-366-1987</li>
3: <li>25 Oak, <i>Forest</i>, Phone (800) 234-7903</li>
4: <li>324 Halsted St., <i>Chicago</i>, Phone 1-<i>800</i>-996-5023</li>
5: <li>700 Lake St., <i>Oak Park</i>, Phone: (708) 798-0008 </li></ul></p>
```

- **Ejemplo:** Extraer el nombre del restaurant “Good Noodles”. Una regla de comienzo (que llamamos R1) puede ser:

R1: *SkipTo()*

Una start rule que significa que el scraper debe comenzar desde el comienzo de la página, y saltar todos los tokens hasta que vea el primer tag. SkipTo() es un operador, y es un landmark.

- De manera similar, para identificar el final del nombre del restaurant, podemos definir:

R2: *SkipTo()*

una end rule que significa “retroceder hasta encontrar </>”, que se ejecuta luego de R1.

Las Reglas no son únicas

- Puede haber varias reglas de identificación para un mismo item. P.ej. podemos usar estas otras reglas para identificar el comienzo de un nombre:

R3: *SkipTo*("Name" _Punctuation_ _HtmlTag_)

o

R4: *SkipTo*("Name") *SkipTo*()

- ***_Punctuation_* y *_HtmlTag_* son comodines (wildcards).**
 - **R3** indica: saltar todo hasta la palabra "Name", seguida de un simbolo de puntuación, y después un tag HTML. En este caso "Name _Punctuation_ _HtmlTag_" todo junto forman un solo landmark.
-

Aprendiendo reglas de extracción

- Stalker utiliza la idea de cobertura secuencial para aprender reglas de extracción para cada item.
 - En cada iteración, aprende una regla perfecta que cubre tantos ejemplos positivos como sea posible sin cubrir ningún ejemplo negativo.
 - Una vez que un ejemplo positivo es cubierto por una regla, se quita de la lista de items a cubrir.
 - El algoritmo termina cuando todos los items positivos fueron cubiertos sin cubrir ningún ejemplo negativo. El resultado es una lista ordenada de reglas de extracción.

El algoritmo general

```
function LearnRules(examples)
  rules = {}
  while examples  $\neq$  {} do:
    disjunct = LearnDisjunct(examples)
    examples = examples - disjunct
    rule = rule  $\cup$  disjunct
  return rules
```

Aprendiendo reglas disjuntas

```
function LearnDisjunct(Examples)
  Seed = ejemplo mas corto en Examples
  Candidates = GetInitialCandidates(Seed)
  while Candidates  $\neq$  {} do:
    D = BestDisjunct(Candidates)
    If D es un disjunto perfecto then
      Return D
    Candidates = Candidates  $\cup$  Refine(D,Seed)
  Candidates = Candidates = D
Return D
```

El prefijo + el ultimo token
del prefijo reemplazado
por wildcards

BestDisjunct() prefiere candidatos
que :

- cubren mas ejemplos correctos
- aceptan menos falsos positivos
- tienen menos wildcards
- tienen landmarks de fin mas largas

Ejemplo: Extraer area codes

Ejemplo de 4 direcciones con código de área anotado:

Ej.1: 205 Willow, <i>Glen</i>, Phone 1-<i>773</i>-366-19873

Ej.2: 25 Oak, <i>Forest</i>, Phone (800) 234-7903

Ej.3: 324 Halsted St., <i>Chicago</i>, Phone 1-<i>800</i>-996-5023

Ej.4: 700 Lake St., <i>Oak Park</i>, Phone: (708) 798-0008

Ejemplo: Extraer código de área dentro de cada dirección

Paso 1: Identificar el conj. de *direcciones*.

Va a identificar el principio y final de las direcciones, p.ej. usando la **start rule** `SkipTo(

)`, y la **end rule** `skipTo(</p>)`, de la siguiente manera:

Iterar sobre la lista (``) para dividir el conjunto en 4 direcciones. Para identificar el comienzo de *c/dirección*, el wrapper puede comenzar desde el 1er. token del nodo padre de las direcciones, y aplicar la **start rule** `skipTo()` muchas veces sobre el contenido de la lista. Cada identificación positiva de comienzo de *address* comenzó donde terminó la identificación del anterior.

Para identificar el final de de cada *address*, el wrapper comienza del ultimo token del parent y aplica la **end rule** `skipTo()`.

Ejemplo: Extraer código de área dentro de cada dirección

Paso 2: Una vez identificado como delimitar *direcciones*:

Para identificar *area_codes* dentro de *direcciones*, sigue de la siguiente manera:

Dado que hay varios formatos de area code, se generan reglas disjuntivas R5 y R6:

start rule R5: skipTo(((or skipTo(-<i>) **end rule R6:** skipTo()) or skipTo(</i>)

En una regla disjuntiva, las disjunciones se aplican secuencialmente hasta que una funciona.

Paso 2, en mas detalle

- Para el ejemplo “Ej.2” de la lista de ejemplos de 4 direcciones, se generan los siguientes reglas candidatas disjuntos:
D1: SkipTo()
D2: SkipTo(*_Punctuation_*)
- D1 es seleccionado por BestDisjunct()
- D1 es un disjunto perfecto.
- La primera iteración de LearnRule() termina. Los ejemplos “Ej.2” y “Ej.4” son quitados de la lista de items a cubrir.

La siguiente iteración del paso 2

- A la próxima iteración para delimitar area codes, le quedan Ej.1 y Ej.3.
- LearnDisjunct() selecciona a Ej.1 como semilla, y genera 2 candidatos :
 - D3: SkipTo(<i>)
 - D4: SkipTo(*_HtmlTag_*) que es la generalización de D3
- Ambos candidatos concuerdan también con ejemplos negativos, en este caso HTML fuera de los items a extraer Ej.1 y Ej.3.
- Como con estas reglas no pueden identificar a los ejemplos positivos solamente, hace falta refinar dichas las reglas.

Refinamiento

- Una regla se especializa agregándole más **terminales**: Un **terminal** es un token o un wildcard que coincida con un token antes (para startRules) o después (para endRules) de la regla hasta ahora.
- La esperanza es que la versión refinada de la regla podrá cubrir algunos ejemplos positivos solamente. sin cubrir ejemplos negativos.
- Stalker tiene 2 tipos de refinamiento de reglas:
 - Landmark refinement
 - Topology refinement

Landmark Refinement

- **Landmark refinement:** Aumentar el tamaño de un landmark concatenándole un terminal.

- P.ej.:

D5: SkipTo(- <i>)

Un nuevo terminal

D6: SkipTo(*Punctuation* <i>)

un wildcard que se corresponde al nuevo terminal

- **Ejemplos:**

D7: SkipTo(205) SkipTo(<i>)

D15: SkipTo(*Numeric*) SkipTo(<i>)

D8: SkipTo(Willow) SkipTo(<i>)

D16: SkipTo(*Alphabetic*) SkipTo(<i>)

D9: SkipTo(,) SkipTo(<i>)

D17: SkipTo(*Punctuation*) SkipTo(<i>)

D10: SkipTo(<i>) SkipTo(<i>)

D18: SkipTo(*HtmlTag*) SkipTo(<i>)

D11: SkipTo(Glen) SkipTo(<i>)

D19: SkipTo(*Capitalized*) SkipTo(<i>)

Topology refinement

- **Topology refinement:** Incrementar la cantidad de landmarks agregando operadores más complejos que se apliquen al token o a su correspondiente generalización (wildcard)
p.ej. Los operadores
- **skipUntil(token_x):** saltar todos los token hasta que ve el token_x, y pararse inmediatamente antes del token. P.ej. skipUntil(Number) salta tokens hasta encontrar un número, y se para inmediatamente antes del número.
- **nextLandmark(sequencia de tokens):** la secuencia de tokens debe seguir inmediatamente a lo que estén antes de la regla. P.ej. SkipTo()nextLandmark() significa que debe aparecer inmediatamente despues de .

La solución entonces es...

- Podemos ver que **D5, D10, D12, D13, D14, D15, D18 y D21** cubren E1 y E3 no cubren E2 and E4.
- Usando **BestDisjunct()**, se selecciona a **D5** como la mejor regla, debido a que tiene el landmark mas largo al final (- <i>).
- Dado que todos los ejemplos fueron cubiertos, LearnRule() retorna la nueva regla de comienzo disjuntiva **R7** (start rule): **D1** o **D5**

R7: *either SkipTo(()*
or SkipTo(- <i>)

Resumiendo a Stalker

- El algoritmo aprende por cubrimiento secuencial.
- Se basa en la idea de landmarks.
- Este algoritmo no es el único posible ni el más eficiente. Hay muchas variaciones y algoritmos completamente diferentes.
- En el ejemplo solo usamos el operador *SkipTo()* , pero puede haber otros, como *SkipUntil()*.

Aprendizaje semi supervisado

- Un scrapper con aprendizaje supervisado necesita ejemplos etiquetados manualmente.
 - Para asegurar un resultado preciso, hacen falta muchos ejemplos etiquetados.
 - Etiquetar ejemplos a mano es caro y laborioso, lo que lo hace difícil de aplicar para un número grande de sitios web.
 - Sería mas útil solo etiquetar aquellos ejemplos que sean necesarios para que el sistema detecte mejor lo que hace falta extraer.
 - Claramente ejemplos que tiene el mismo formateo que ejemplos ya etiquetados son de poca utilidad para diferenciar casos positivos de negativos.
 - Los ejemplos que representan excepciones son más útiles, debido a que son diferentes a otros ejemplos ya etiquetados.
-

Active learning

-
- Active learning es una interacción entre el algoritmo y el usuario para identificar ejemplos informativos aún sin etiquetar que mejoren el aprendizaje del algoritmo.
-
1. Elegir al azar un subconjunto L *pequeño* de ejemplos sin etiquetar de U (*el total de ejemplos*)
 2. Etiquetar manualmente a los ejemplos en L , y hacer $U = U - L$
 3. Aprender un scrapper S basado en los ejemplos en L
 4. Aplicar S a U para encontrar un conjunto de ejemplos L que sea informativo.
 5. Volver a empezar si L no quedó vacío.
-

Identificando ejemplos informativos

- Co-testing toma ventaja del hecho que frecuentemente hay varias maneras de extraer el mismo item.
 - Así, el sistema puede aprender tanto **forward** y **backward rules** para identificar al mismo item.
 - Las reglas aprendidas en el ejemplo de Stalker se llaman *forward rules*, porque consumen tokens desde el comienzo del texto hasta el fin.
 - De manera similar , podemos aprender *backward rules* que consuman tokens desde el final del texto hacia el comienzo.
-

Identificando ejemplos informativos

- Dado un ejemplo sin etiquetar, se infieren tanto reglas hacia adelante como hacia atrás.
- **Intuición:** Cuando las 2 reglas coinciden, la extracción es probablemente correcta.
- Cuando una de las reglas no está de acuerdo con la otra, una debe estar mal.
- Si 2 reglas están en desacuerdo en el comienzo de un item, el ejemplo es mostrado al usuario para que lo identifique manualmente, y así mejorar el aprendizaje.

Mantenimiento de Scrappers

- **Verificación:** Si el sitio cambia, el scrapper ¿es capaz de identificar el cambio?
 - **Reparación:** Si el scrapper identifica correctamente el cambio, ¿cómo reparar automáticamente al scrapper?
 - Estas tareas son difíciles porque hace falta entender el significado (semántica) de los resultados para detectar cambios y encontrar la nueva ubicación de los items.
 - Una manera de enfrentar estos problemas es identificar reglas sobre la forma correcta de los items a extraer (p.ej. Expresiones regulares que cumplan todos ejemplos positivos ya extraídos)
 - Estos patrones se usan para monitorear lo que el scrapper entrega.
 - Mantener scrappers es siempre costoso, porque la web es un ambiente dinámico donde los sitios cambian constantemente.
-

Portia, un scrapper open source con GUI

Portia (inducción automática de scrappers, sin usar xpaths)

<https://github.com/scrapinghub/portia>

The screenshot displays the Portia GUI interface for configuring a web scraper project. The left sidebar contains a 'Commands' panel with a tree view of the project structure, including 'main_template', 'Select page', 'Select song', 'Begin new entry in song', 'Extract name', 'Relative artist', 'Relative position', 'Extract position', 'Hover', 'Relative last_week', 'Relative peak', and 'Relative weeks_on_chart'. Below this is a 'Selection Node' section with a dropdown menu set to 'All elements' and a checkbox for 'Wait up to 60 seconds for elements to appear'. A 'Get Data' button is at the bottom of the sidebar. The main area shows a preview of the 'THE HOT 100' chart from Spotify, with a search bar for 'ARCHIVE SEARCH' and a date selector for 'JULY 9, 2016'. The chart lists songs like 'One Dance' by Drake, 'Can't Stop The Feeling!' by Justin Timberlake, and 'Panda' by Deshaun. A 'Test Run' button is visible at the bottom of the main area.

Home > Projects > * www.billboard.com Project

Commands Settings SELECT

main_template Options

Select page +

Select song +

Begin new entry in song +

Extract name +

Relative artist +

Relative position (100)

Extract position

Hover

Relative last_week +

Relative peak +

Relative weeks_on_chart +

Selection Node: Edit

> All elements

All elements with class chart-row__current-week

☐ Wait up to 60 seconds for elements to appear.

help tips ?

Get Data

Help Center | Contact us

THE HOT 100

Listen exclusively with Spotify

How it works

THE WEEK OF JULY 9, 2016

ARCHIVE SEARCH MM/DD/YYYY

LAST WEEK: 1 PEAK POSITION: 1 WKS ON CHART: 12

Gains in performance

1 One Dance Drake Featuring WizKid & Kyla Full Track

2 Can't Stop The Feeling! Justin Timberlake Video

3 Panda Deshaun

Sample is disabled when working on interactive templates. Use Test Run instead.

Visuals enabled

CENTENNIAL COLLEGE AVIATION TECHNICIAN 1 of over 250 programs. Discover Our Programs