

RECUPERACIÓN DE INFORMACIÓN Y WEB SEMÁNTICA

Tecnologías y Proyecto



September 2024

Javier Parapar

✉ javier.parapar@udc.es

Anxo Pérez

✉ anxo.pvila@udc.es

Information Retrieval Lab
Computer Science Department
University of A Coruña

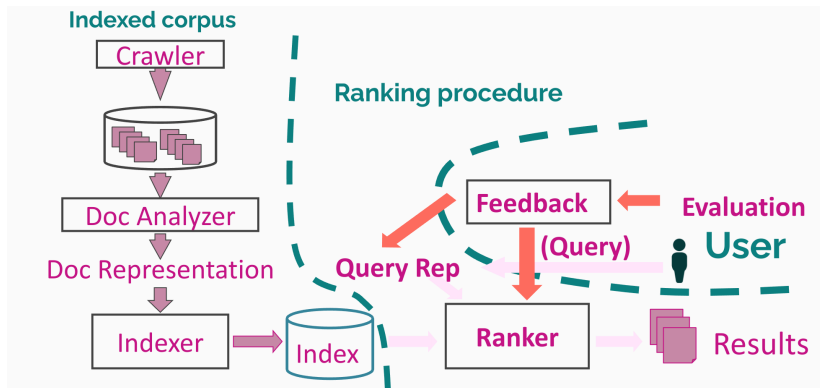


Introducción

- El problema central en la Recuperación de Información es encontrar, en una colección, aquellos documentos que son relevantes para la necesidad de información del usuario.
 - Documento: generalmente texto no estructurado, pero puede ser cualquier otro recurso, como imágenes, videos, música, etc.
 - Colección: gran conjunto de datos (repositorios, sistemas de archivos, la Web, etc.)
- **IR:** Área de la informática responsable de desarrollar técnicas y modelos para facilitar la búsqueda de documentos en grandes colecciones para satisfacer una necesidad de información del usuario.

- 1. **Crawling**: Recolección de datos de diversas fuentes.
- 2. **Indexación**: Los datos son procesados y almacenados en un índice para facilitar la búsqueda.
- 3. **Consultas**: El usuario realiza una consulta para encontrar información relevante.
- 4. **Matching**: Se comparan las consultas con los documentos indexados para encontrar coincidencias.
- 5. **Ranking**: Los resultados se clasifican según relevancia.

Arquitectura de un motor de búsqueda



- **Documentos:** La fuente de información a recuperar (textos, imágenes, videos, etc.).
- **Consulta:** Lo que el usuario ingresa en el sistema (generalmente en forma de palabras clave).
- **Índice:** Estructura de datos que permite el acceso rápido a la información almacenada.
- **Modelo de Recuperación:** Algoritmo que compara las consultas con los documentos.
- **Ranking:** Criterio para ordenar los resultados en función de su relevancia.

- **Modelo Booleano:** Usa operadores lógicos para coincidir consultas y documentos.
- **Modelo Vectorial:** Representa consultas y documentos como vectores en un espacio multidimensional.
- **Modelo Probabilístico:** Calcula la probabilidad de relevancia de cada documento con respecto a la consulta.
- **TF-IDF:** Usa la frecuencia de términos y su rareza en la colección para ponderar la relevancia.

- **Ambigüedad de las consultas:** Las palabras tienen significados múltiples.
- **Escalabilidad:** Los sistemas deben manejar grandes volúmenes de datos.
- **Evaluación:** Medir la relevancia de los resultados puede ser subjetivo.
- **Búsqueda semántica:** Entender el significado de la consulta más allá de las palabras clave -> (Embeddings + LLMs!)

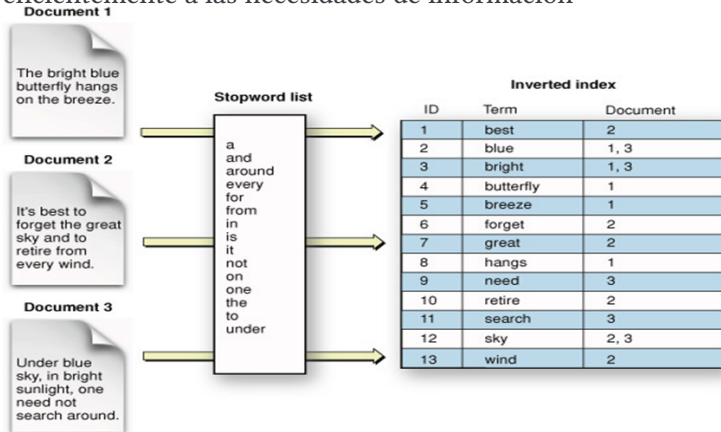
¿Qué es un Índice Invertido?

- Un **Índice Invertido** es una estructura de datos utilizada para mapear el contenido de documentos a posiciones específicas dentro de ellos.
- Es la base de los motores de búsqueda para realizar consultas eficientes y resolver el problema de la **escalabilidad**.
- Permite recuperar rápidamente todos los documentos que contienen una palabra específica.

- El índice invertido se compone de:
 - **Término**: La palabra clave que se ha indexado.
 - **Lista de Documentos**: Un conjunto de documentos que contienen el término.
 - Opcionalmente, se incluyen **posiciones** o **frecuencia** de cada término en cada documento.
- Este índice permite una búsqueda más rápida en comparación con la búsqueda secuencial.

¿Qué es un Índice Invertido?

- Los **índices invertidos** fueron desarrollados para responder eficientemente a las necesidades de información



- Una vez que se han construido los índices invertidos, nos permiten obtener todos aquellos documentos que contienen los términos de la consulta del usuario simplemente operando sobre las *listas de publicaciones*.
- Dependiendo del modelo de recuperación y los esquemas de ponderación que queramos usar, a los documentos se les asignará una puntuación determinada por la fórmula de puntuación del modelo: $\text{sim}(Q,D)$
- Los documentos serán ordenados posteriormente en orden descendente según dicha puntuación. Al usuario se le presenta la lista de documentos en forma de un ranking en respuesta a su consulta.

Por supuesto no es tan simple...

- **TF-IDF** means Term Frequency * Inverse Document Frequency
- **Term Frequency** is how often a term appears in a given document
- **Document Frequency** is how often a term appears in all documents
- Term Frequency / Document Frequency measures the **relevance** of a term in a document

ElasticSearch



- **ElasticSearch** es un motor de búsqueda y análisis distribuido, de código abierto, basado en Apache Lucene.
- Permite realizar búsquedas y analizar grandes volúmenes de datos de manera rápida y en tiempo real.
- Es parte del **Elastic Stack** (también conocido como ELK Stack), junto con Logstash, Kibana y Beats.

- **Motores de búsqueda internos:** Para búsqueda rápida y eficiente dentro de aplicaciones web y plataformas.
- **Monitoreo y análisis de logs:** Herramientas como Kibana, junto con Elasticsearch, permiten analizar grandes cantidades de logs en tiempo real.
- **Análisis de datos empresariales:** Usado para realizar búsquedas y análisis en grandes datasets empresariales.
- **Búsqueda de texto completo:** Utilizado en aplicaciones donde la búsqueda rápida de documentos y texto es crucial.
- **Recomendación de productos:** Empresas de comercio electrónico lo utilizan para crear sistemas de recomendación en tiempo real.

- **Escalabilidad:** Elasticsearch está diseñado para ser escalable horizontalmente, permitiendo añadir más nodos fácilmente.
- **Búsqueda en tiempo real:** Proporciona resultados de búsqueda casi instantáneos, incluso con grandes volúmenes de datos.
- **Distribuido:** Los datos están distribuidos entre varios nodos, lo que mejora la tolerancia a fallos y la disponibilidad.
- **Soporte para búsquedas complejas:** Incluye búsqueda por texto completo, consultas booleanas, agregaciones y filtros.
- **RESTful API:** Facilita la integración con otras aplicaciones a través de su API basada en HTTP y JSON-based.



WIKIPEDIA
La enciclopedia libre



- **Netflix:** Utiliza Elasticsearch para el monitoreo y análisis en tiempo real de sus sistemas de streaming y datos de usuarios.
- **Uber:** Lo emplea para monitorear eventos en tiempo real y optimizar las rutas de los conductores.
- **Slack:** Elasticsearch es clave en su sistema de búsqueda interno para ofrecer búsquedas rápidas entre los mensajes de los usuarios.
- **Shopify:** Usado para mejorar la experiencia de búsqueda y recomendaciones en su plataforma de comercio electrónico.
- **Wikipedia:** Implementa Elasticsearch para manejar la búsqueda de artículos en toda la plataforma, procesando grandes volúmenes de texto.

- Un **Índice** en Elasticsearch es una colección de documentos relacionados que comparten características comunes.
- Se puede comparar con una base de datos en sistemas de bases de datos tradicionales.
- Los índices están identificados por un nombre, y este nombre es utilizado para realizar búsquedas, actualizaciones y borrados de documentos.
- Cada índice está dividido en **shards**, lo que permite distribuir los datos en varios nodos.

- Un **Documento** es la unidad básica de almacenamiento de información en Elasticsearch.
- Cada documento es un objeto en formato JSON que contiene datos organizados en **campos**.
- Los documentos pueden ser muy flexibles y contener cualquier tipo de información, como datos de productos, logs, usuarios, etc.

- Elasticsearch permite tener documentos de diferentes **tipos** dentro de un índice.
- Los documentos de diferentes tipos pueden tener diferentes estructuras, pero todos pertenecen al mismo índice:
 - **Producto**: Contiene información sobre un producto (nombre, precio, descripción, etc.).
 - **Usuario**: Almacena datos sobre usuarios (nombre, email, fecha de registro, etc.).
 - **Pedido**: Representa un pedido con campos como fecha, monto, artículos, etc.

- Cada documento está compuesto por **campos**, que son las unidades individuales de datos dentro del documento.
- Un campo tiene un **nombre** y un **valor**. El valor puede ser de varios tipos, como:
 - Texto
 - Números (enteros, decimales)
 - Fechas
 - Booleanos
- Los campos permiten hacer búsquedas avanzadas y específicas en Elasticsearch.

```
{  
  "producto_id": 123,  
  "nombre": "Camiseta",  
  "precio": 25.99,  
  "disponible": true,  
  "descripcion": "Camiseta de algodón, talla M",  
  "fecha_agregado": "2024-01-01"  
}
```

- Este documento representa un producto con varios campos como **producto_id**, **nombre**, **precio**, etc.
- Cada campo es clave para permitir búsquedas eficientes en Elasticsearch.

Playing with Elasticsearch

- Descargamos Elasticsearch desde la página oficial:
 - `https://www.elastic.co/downloads/elasticsearch`
- Elegir la versión adecuada según tu sistema operativo:
 - Linux, macOS: Archivo .tar.gz o .deb.
 - Windows: Archivo .zip o instalador MSI.
- El archivo descargado contendrá los binarios necesarios para ejecutar Elasticsearch.

- Para desactivar la autenticación, y poder conectarte a Elastic (puerto 9200) sin necesidad de usuario/contraseña
 - Ir al fichero de configuración *elasticsearch.yml* y setear:
xpack.security.enabled: false y
xpack.security.enrollment.enabled: false
- Uso con python:

```
pip install elasticsearch
```

Creación de un Índice en Elasticsearch

```
create_index.py > ...
1  from elasticsearch import Elasticsearch
2
3  # Conexión con Elasticsearch en localhost
4  es = Elasticsearch(hosts=["http://localhost:9200"])
5
6  # Nombre del índice
7  index_name = "productos"
8
9  # Define el mapeo de campos (estructura del documento)
10 mapping = {
11     "mappings": {
12         "properties": {
13             "nombre": {"type": "text"},
14             "precio": {"type": "float"},
15             "disponible": {"type": "boolean"}
16         }
17     }
18 }
19
20 # Crea el índice
21 es.indices.create(index=index_name, body=mapping)
22 print(f"Índice '{index_name}' creado con éxito.")
23
```

- Creamos un índice llamado **productos** con un **mapping** que define tres campos: nombre (texto), precio (float) y disponible (booleano).

Inserción de Documentos en Elasticsearch

```
Insert_docs.py > ...
1  from elasticsearch import Elasticsearch
2
3  # Conexión con Elasticsearch en localhost
4  es = Elasticsearch(hosts=["http://localhost:9200"])
5
6
7  # Documentos a indexar
8  doc1 = {
9      "nombre": "Camiseta",
10     "precio": 19.99,
11     "disponible": True
12 }
13
14  doc2 = {
15     "nombre": "Pantalones",
16     "precio": 49.99,
17     "disponible": False
18 }
19
20 # Inserta los documentos en el índice
21 es.index(index="productos", id=1, body=doc1)
22 es.index(index="productos", id=2, body=doc2)
```

- Insertamos dos documentos en el índice productos, asignando un ID a cada documento.

Búsqueda de Documentos en Elasticsearch

```
search_docs.py > ...
1  from elasticsearch import Elasticsearch
2
3  # Conexión con Elasticsearch en localhost
4  es = Elasticsearch(hosts=["http://localhost:9200"])
5
6  # Definimos la consulta de búsqueda
7  simple_query = {
8      "query": {
9          "match": {
10              "nombre": "Camiseta"
11          }
12      }
13  }
14
15  # Ejecutar la búsqueda
16  res = es.search(index="productos", body=simple_query)
17
18  # Imprimir los resultados de la búsqueda
19  print("Resultados de la búsqueda:")
20  for hit in res['hits']['hits']:
21      print(hit["_source"])
```

- Buscamos todos los documentos cuyo campo nombre contenga la palabra Camiseta.
- Los resultados se imprimen mostrando los documentos coincidentes en el índice.

Búsqueda Avanzada con Filtros

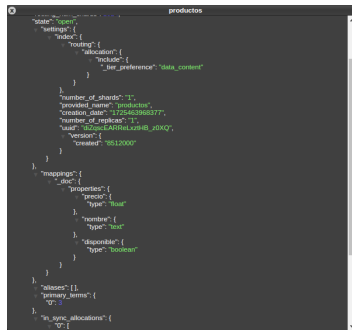
```
25 # Definimos la búsqueda con filtros
26 range_query = {
27     "query": {
28         "bool": {
29             "must": [
30                 {"match": {"disponible": True}}
31             ],
32             "filter": [
33                 {"range": {"precio": {"lte": 30}}}
34             ]
35         }
36     }
37 }
38
39 # Ejecutar la búsqueda
40 res = es.search(index="productos", body=range_query)
41
42 # Imprimir los resultados de la búsqueda
43 print("Resultados de la búsqueda con filtros:")
44 for hit in res['hits']['hits']:
45     print(hit["_source"])
```

- Filtramos los productos disponibles con un precio menor o igual a 30 euros.
- Usa una combinación de una consulta booleana (`bool`) para agregar varios criterios: `must` (condiciones) y `filter` (filtros).

ElasticSearch Head



- Elasticsearch-head es una interfaz gráfica basada en la web para interactuar con un clúster de Elasticsearch
- <http://mobz.github.io/elasticsearch-head/>



```

{
  "state": "open",
  "settings": {
    "index": {
      "routing": {
        "allocation": {
          "include": {
            "_tier_preference": "data_content"
          }
        }
      },
      "number_of_shards": "1",
      "provided_name": "productos",
      "creation_date": "1725463968377",
      "number_of_replicas": "1",
      "uuid": "d829dc6ARRRkLx9tB_r0XQ",
      "version": {
        "created": "8512000"
      }
    }
  },
  "mappings": {
    "_doc": {
      "properties": {
        "precio": {
          "type": "float"
        },
        "nombre": {
          "type": "text"
        },
        "disponible": {
          "type": "boolean"
        }
      }
    }
  },
  "aliases": [],
  "primary_terms": {
    "0": {}
  },
  "in_sync_allocations": {
    "0": {}
  }
}
```

- Te permite ver la estructura de los índices, como el mapeo de los campos, y analizar cómo se almacenan los datos.



The screenshot shows the ElasticSearch Head web interface. At the top, there's a header with the title 'Elasticsearch' and the URL 'http://localhost:9200/'. Below this, there are tabs for 'Overview', 'Indices', 'Browser', 'Structured Query [+]', and 'Any Request [+]', with 'Indices' being the active tab. The main area shows a search query: 'Search: productos (2 docs) for documents where: must [match_all]'. Below the query, there's a 'Search' button and a 'Show query source' checkbox. The results are displayed in JSON format, showing two documents. The first document is a 'producto' with a score of 1, and the second document is a 'producto' with a score of 1. The JSON output is as follows:

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "productos",
        "_id": "1",
        "_score": 1,
        "_source": {
          "nombre": "Camiseta",
          "precio": 10.00,
          "disponible": true
        }
      },
      {
        "_index": "productos",
        "_id": "2",
        "_score": 1,
        "_source": {
          "nombre": "Pantalones",
          "precio": 40.00,
          "disponible": false
        }
      }
    ]
  }
}
```

- Te permite escribir y ejecutar consultas de búsqueda sobre los índices.

Scrapy



- **Scrapy** es un framework de código abierto en Python diseñado para realizar **web scraping** y **crawling web**.
- Facilita la extracción de información de sitios web y la estructura en datos organizados.
- Permite navegar y extraer datos automáticamente de grandes cantidades de páginas web.
- Soporta tareas complejas como seguir enlaces, manejar páginas dinámicas y trabajar con APIs.

- **Scraping de Productos:** Extraer datos de productos en tiendas online (precio, descripciones, reseñas).
- **Agregación de Noticias:** Recopilar noticias de diferentes fuentes para crear un agregador de noticias.
- **Monitoreo de Precios:** Rastrear y monitorear cambios de precios en e-commerce.
- **Análisis de Competencia:** Obtener información de competidores como catálogos de productos, estrategias de marketing, etc.
- **Extracción de Datos para Machine Learning:** Usado para recopilar datasets a partir de información disponible en sitios web.

- **Zillow**: Utiliza Scrapy para rastrear anuncios de propiedades y analizar el mercado inmobiliario.
- **Lyst**: Utiliza Scrapy para recopilar datos de productos de moda de diversas tiendas online.
- **Yelp**: Scrapy es empleado para recopilar reseñas y comentarios de usuarios.
- **Eventbrite**: Utiliza Scrapy para extraer información sobre eventos y entradas en distintos lugares.

Playing with Scrapy

- Fácil instalación usando pip:

```
pip install scrapy
```

- Una vez instalado, puedes verificar la instalación ejecutando:

```
scrapy -h
```


Creación de un Proyecto Scrapy

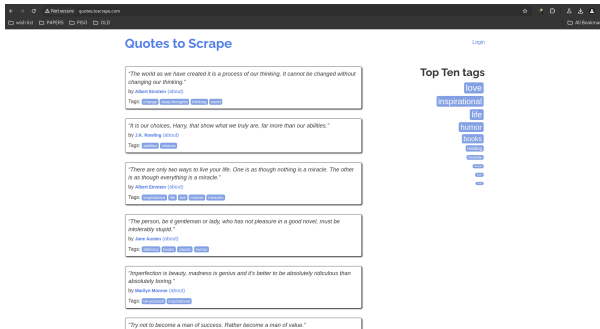
- Tutorial + doc de scrapy: <https://docs.scrapy.org/en/latest/intro/tutorial.html>

- Para crear un proyecto básico en scrapy:

```
scrapy startproject nombre_proyecto
```

- Esto creará una estructura de carpetas con:
 - **spiders/**: Aquí se definen los scripts para el crawling.
 - **items.py**: Define las estructuras de los datos a extraer.
 - **settings.py**: Configuraciones del proyecto Scrapy.
 - **pipelines.py**: Gestiona el procesamiento posterior de los datos extraídos, como limpieza, validación o almacenamiento en bases de datos, después de que el spider los haya recopilado.

Ejemplo de dominio a crawl



- Queremos:
 - Obtener la información de cada quote con la siguiente información: texto de la **quote, autores, y tags**.
 - Guarda todo esto en un archivo **JSON**.

[Login](#)

by Albert Einstein (about)

Tags: [change](#) [deep-thoughts](#) [thinking](#) [world](#)

by J.K. Rowling (about)

Tags: [abilities](#) [choices](#)

"There are only two ways to live your life. One is as though nothing is

```

<div class="container">
  <!--before-->
  <div class="row header-box"></div>
  <div class="row">
    <!--before-->
    <div class="col-md-8">
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"> == 80
        <span class="text" itemprop="text"></span>
        <!--<span></span>-->
        <div class="tags"></div>
      </div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <div class="quote" itemscope itemtype="http://schema.org/CreativeWork"></div>
      <nav></nav>
    </div>
    <div class="col-md-4 tags-box"></div>
    <!--after-->
  </div>
  <!--after-->
</div>
<div class="footer"></div>
</body>
</html>

```

Spider de Quotes

```
crawling-project > intro_rnws > intro_rnws > spiders > quotes_spider.py > QuotesSpider
1  import scrapy
2  from intro_rnws.items import QuoteItem # Import the QuoteItem class
3
4  class QuotesSpider(scrapy.Spider):
5      name = "quotes"
6
7      # Start URL
8      start_urls = [
9          'http://quotes.toscrape.com/'
10     ]
11
12     # Parse method to extract quotes
13     def parse(self, response):
14         for quote in response.css('div.quote'):
15             # Create an instance of QuoteItem
16             item = QuoteItem()
17             item['text'] = quote.css('span.text::text').get()
18             item['author'] = quote.css('span small::text').get()
19             item['tags'] = quote.css('div.tags a.tag::text').getall()
20
21             yield item # Yield the item for the pipeline to process
22
23             # Follow the next page link if available
24             next_page = response.css('li.next a::attr(href)').get()
25             if next_page is not None:
26                 next_page = response.urljoin(next_page)
27                 yield scrapy.Request(next_page, callback=self.parse)
28
```

- **scrapy.Spider**: La clase QuotesSpider hereda de scrapy.Spider, definiendo un spider para realizar el crawling en el sitio web `http://quotes.toscrape.com/`.
- **start_urls**: Este atributo define la lista de URLs de inicio. En este caso, el spider comienza a rastrear desde la página principal de citas.
- **Método parse**: Es el método principal que Scrapy llama para procesar las respuestas de las solicitudes. En este caso, usamos selectores CSS (`response.css()`) para extraer citas, autores y etiquetas de cada cita dentro de la página.

- **QuoteItem**: Se crea una instancia de la clase QuoteItem (definida en items.py) para representar los datos extraídos de cada cita.

Enlaces a la siguiente página: Verifica si hay un enlace de paginación (next_page) y, si lo encuentra, hace una solicitud para la siguiente página, llamando de nuevo al método parse recursivamente para seguir extrayendo datos.

```
quotes_spider.py × items.py ×
crawling-project > intro_r1ws > intro_r1ws > items.py > ...
1  # Define here the models for your scraped items
2  #
3  # See documentation in:
4  # https://docs.scrapy.org/en/latest/topics/items.html
5
6  import scrapy
7
8  class QuoteItem(scrapy.Item):
9      text = scrapy.Field()
10     author = scrapy.Field()
11     tags = scrapy.Field()
12
```

- **items.py**: Define las estructuras de los datos a extraer.


```
quotes_spider.py × pipelines.py ×
crawling-project > intro_r1ws > intro_r1ws > pipelines.py > JsonWriterPipeline > process_item
1 import json
2
3 class JsonWriterPipeline:
4     def open_spider(self, spider):
5         self.file = open('quotes.json', 'w')
6         self.file.write('[')
7         self.first_item = True # Flag to check if this is the first item
8
9     def close_spider(self, spider):
10        self.file.write(']\n')
11        self.file.close()
12
13    def process_item(self, item, spider):
14        if not self.first_item:
15            self.file.write(',')
16            self.first_item = False
17            line = json.dumps(dict(item), indent=4)
18            self.file.write(line)
19        return item
20
```

- **Clase JsonWriterPipeline:** Este pipeline gestiona el proceso de escribir los elementos extraídos por el spider en un archivo JSON "quotes.json".
- **Método open_spider:** Se ejecuta cuando el spider comienza a ejecutarse.
- **Método close_spider:** Se ejecuta cuando el spider finaliza su ejecución.
- **Método process_item:** Se llama cada vez que el spider extrae un nuevo elemento.

```
crawling-project > intro_r1ws > intro_r1ws > settings.py > ...
1 # Scrapy settings for intro_r1ws project
2 #
3 # For simplicity, this file contains only settings considered important or
4 # commonly used. You can find more settings consulting the documentation:
5 #
6 # https://docs.scrapy.org/en/latest/topics/settings.html
7 # https://docs.scrapy.org/en/latest/topics/downloader-middleware.html
8 # https://docs.scrapy.org/en/latest/topics/spider-middleware.html
9
10 BOT_NAME = "intro_r1ws"
11
12 SPIDER_MODULES = ["intro_r1ws.spiders"]
13 NEWSPIDER_MODULE = "intro_r1ws.spiders"
14
15
16 # Crawl responsibly by identifying yourself (and your website) on the user-agent
17 #USER_AGENT = "intro_r1ws (+http://www.yourdomain.com)"
18
19 # Obey robots.txt rules
20 ROBOTSTXT_OBEY = False
21
22 ITEM_PIPELINES = {
23     'intro_r1ws.pipelines.JsonWriterPipeline': 1, # Enable the pipeline
24 }
```

- **BOT_NAME**: Define el nombre del bot de Scrapy.
- **SPIDER_MODULES**: Especifica los módulos donde Scrapy buscará los spiders.
- **NEWSPIDER_MODULE**: Define dónde se crearán los nuevos spiders cuando se ejecuta el comando scrapy genspider.
- **ROBOTSTXT_OBEY**: Configura si el bot debe obedecer las reglas definidas en el archivo robots.txt de los sitios web.
- **ITEM_PIPELINES**: Define qué pipelines se usarán y en qué orden. El diccionario contiene el pipeline JsonWriterPipeline, asignándole una prioridad de 1.

- Cómo lanzamos el Spider?:

```
scrapy crawl [nombre_spider]
```

Apache Nutch



- Es un *web crawler* extensible y escalable.
- Proporciona entre otras cosas:
 1. *Web crawling*.
 2. WebGrah y LinkRank.
 3. Detección de formatos de documento y parsing.
 4. Detección de lenguaje, y codificación.
 5. Mecanismo de extensión mediante plugins.
 6. Soporte de múltiples *backends* de persistencia.
- Originalmente Hadoop nació como parte de Nutch.

- Hay dos ramas de Nutch soportadas:
 - Nutch 1.x: versión madura. Usa estructuras de datos de Hadoop (HDFS).
 - Nutch 2.x: versión más moderna. Emplea Apache Gora para gestionar la persistencia. En concreto, Nutch 2.4 usa Gora 0.8 que soporta:
 - Apache Avro 1.8.1
 - Apache Hadoop 2.5.2
 - Apache HBase 1.2.3
 - Apache Cassandra 3.11.0 (Datastax Java Driver 3.3.0)
 - Apache Solr 6.5.1
 - MongoDB (driver) 3.5.0
 - Apache Accumlo 1.7.1
 - Apache Spark 1.4.1
 - Apache CouchDB 1.4.2 (test containers 1.1.0)
 - Amazon DynamoDB (driver) 1.10.55
 - Infinispan 7.2.5.Final
 - JCache 1.0.0 with Hazelcast 3.6.4 support.
 - OrientDB 2.2.22
 - Aerospike 4.0.6

- Nos bajamos los fuentes:

```
wget https://downloads.apache.org/nutch/2.4/apache-nutch-2.4-src
```

- Configuramos Nutch en `conf/nutch-site.xml`. En `conf/nutch-default.xml` tenemos los valores por defecto. Debemos establecer al menos el *agent name*:

```
<property>  
<name>http.agent.name</name>  
<value>My RIWS Spider 1.0</value>  
</property>
```

Instalando Nutch 2.4 (II)

- Mostramos cómo usar MongoDB como *backend*, pero configurar otro sistema sería similar.
- Añadir el *backend* en `conf/nutch-site.xml`:

```
<property>
<name>storage.data.store.class</name>
<value>org.apache.gora.mongodb.store.MongoStore</value>
<description>Class for storing data</description>
</property>
```

- Descomentar el *backend* en `ivy/ivy.xml` y ajustar la versión adecuada del mismo:

```
<dependency org="org.apache.gora" name="gora-mongodb" rev="0.8" conf="*->default" />
```

- Añadir el *backend* en `conf/gora.properties`:

```
gora.mongodb.override_hadoop_configuration=false
gora.mongodb.mapping.file=/gora-mongodb-mapping.xml
gora.mongodb.servers=localhost:27017
gora.mongodb.db=nutch-riws
```

- Compilamos el código con:

`ant runtime`

- Si os falla la compilación porque no encuentra unas dependencias hay que añadir un repo en el *ivy/ivysettings.xml*

```
<property name="repo.restlet"  
value="https://maven.restlet.talend.com/"  
override="false"/>
```

```
...
```

```
<resolvers>  
<ibiblio name="restlet"  
root="${repo.restlet}"  
pattern="${maven2.pattern.ext}"  
m2compatible="true"  
</>  
...
```

```
<chain name="default" dual="true">  
<resolver ref="local"/>  
<resolver ref="maven2"/>  
<resolver ref="sonatype"/>  
<resolver ref="apache-snapshot"/>  
<resolver ref="spring-plugins"/>  
<resolver ref="restlet"/>  
</chain>  
...
```

- Descargamos Mongo:

```
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1604-3.4.7.tgz  
tar xzvf mongodb-linux-x86_64-ubuntu1604-3.4.7.tgz
```

- Creamos las carpetas de data y logs:

```
mkdir data logs
```

- Arrancamos Mongo:

```
./mongodb-linux-x86_64-ubuntu1604-3.4.7bin/mongod --dbpath data/ \  
--logpath logs/mongo-riws.log
```

- Eliminamos el managed-schema de Solr y opiamos el `schema.xml` de Nutch en la colección de Solr correspondiente.

```
rm solr-8.4.1/server/solr/nutch-riws/conf/managed-schema
cp apache-nutch-2.4/conf/schema.xml
solr-8.4.1/server/solr/nutch-riws/conf/
```

- Eliminamos las ocurrencias de `enablePositionIncrements="true"` en el `schema.xml` que copiamos a la configuración del core.
- Eliminamos `<solrQueryParser defaultOperator="OR"/>` y `<defaultSearchField>text</defaultSearchField>`
- Comentamos el bloque de `<updateProcessor class="solr.AddSchemaFields" name="add-schema-fields">...` en:
`solr-8.4.1/server/solr/nutch-riws/conf/solrconfig.xml` y eliminamos `add-schema-field` de la chain `"add-unknown-fields-to-the-schema"`

- Arrancamos Solr y comprobamos que funciona y levanta el core:
`http://localhost:8983/solr`
- Añadimos el plugin `indexer-solr` usando la propiedad *plugin.includes* modificando el fichero de configuración:
runtime/local/conf/nutch-site.xml:

```
<property>
<name>plugin.includes</name>
<value>protocol-http|urlfilter-regex|
parse-(html|tika)|index-(basic|anchor)|
urlnormalizer-(pass|regex|basic)|
scoring-opic|indexer-solr</value>
</property>
```

- Vamos a *crawlear* solo webs que cuelguen de `https://www.fic.udc.es`. Para ello debemos editar el fichero `runtime/local/conf/regurlfilter.txt` (indica qué URLs se pueden explorar) y cambiamos la última línea por:

```
+^https://www.fic.udc.es
```

- Añadimos las URLs semilla de nuestro *crawling*:

```
mkdir runtime/local/urls  
echo "https://www.fic.udc.es/" > \  
runtime/local/urls/seed.txt
```

- Ahora iniciamos el *crawling* del sitio web:

```
cd runtime/local  
bin/nutch inject urls  
bin/nutch generate -topN 25  
bin/nutch fetch -all  
bin/nutch parse -all  
bin/nutch updatedb -all
```

- Se pueden repetir los últimos cuatro comandos para profundizar en el *crawling*.

- Para pasar los datos de HBase a Solar:

```
bin/nutch solrindex http://localhost:8983/solr/nutch-riws -all
```

- Para consultar los datos importados:

```
http://localhost:8983/solr/#/nutch-riws/query
```

- Para definir nuevos campos en el *crawling* de Nutch o nuevas reglas de *parsing*, crearemos un plugin. Para indexar los nuevos campos en Solr, hay que editar el `schema.xml`. Tutorial con un ejemplo para almacenar información de los metatags de las webs:

```
http://wiki.apache.org/nutch/IndexMetatags
```

- Véase también la clase:

```
src/plugin/index-metadata/src/java/org/apache/nutch/  
indexer/metadata/MetadataIndexer.java
```


- Wiki de Nutch:
`https://wiki.apache.org/nutch`
- Tutorial de Nutch 2:
`http://wiki.apache.org/nutch/Nutch2Tutorial`
- Nutch ahora ofrece una interfaz web autocontenida:
`https://issues.apache.org/jira/browse/NUTCH-841`
- Documentación del API REST de Nutch:
`https://wiki.apache.org/nutch/NutchRESTAPI`

- Cuando haya algún problema consultad los logs. Todos los proyectos Apache tienen una carpeta logs.
- Si Nutch da problemas, comprobad que el *backend* de almacenamiento está funcionando correctamente.
- Si no somos capaces de *crawlear* un sitio web, comprobad que no tenemos problemas de conexión y que la web no nos está bloqueando. Puede ser útil cambiar el *useragent* de Nutch por otro de un navegador común, por ejemplo:

Mozilla/5.0 (X11; Linux x86_64; rv:42.0) Gecko/20

Proyecto

- Definir un **dominio de recogida de información** y definir una **estrategia de crawling web** (Para ese dominio, se pueden usar una, dos o las web que hagan falta!!).
- Definir una **tarea específica búsqueda** sobre ese dominio (p. ej., *ad hoc retrieval* o *web retrieval*).
- **Postprocesar los resultados** (ir más allá de una mera lista de enlaces). Algunos ejemplos:
 - Clustering de resultados de búsqueda.
 - Permitir que el usuario refine los resultados de búsqueda a través de filtros basados en categorías
 - Recomendación de items similares a su búsqueda
 - Generación de sumarios o eliminación de documentos redundantes
 - Personalización basada en idioma, geolocalización...
 - Visualización de resultados con gráficas o diagramas

- **Definir un interfaz** de interacción y de visualización de resultados adecuado.
- **Presentar propuesta para aprobación** antes de implementarlo

- ¿Quién usa Lucene?
<https://wiki.apache.org/lucene-java/PoweredBy>
- ¿Quién usa Solr?
<https://wiki.apache.org/solr/PublicServers>

- Fecha límite de entrega: 15/11/2024
- Grupos: de hasta 3 personas
- Entregables:
 - Código fuente
 - Binarios/ejecutables/empaquetado
 - Memoria de la práctica. Debe incluir información sobre:
 - Búsqueda del dominio y crawling web.
 - Desarrollo del proyecto.
 - Funcionalidades implementadas.
 - Tecnologías usadas.

Gracias!