Information Retrieval Project
Youtube Search Engine

Stefano Peverelli, Matteo Piergiovanni

December 2015

Problem Definition

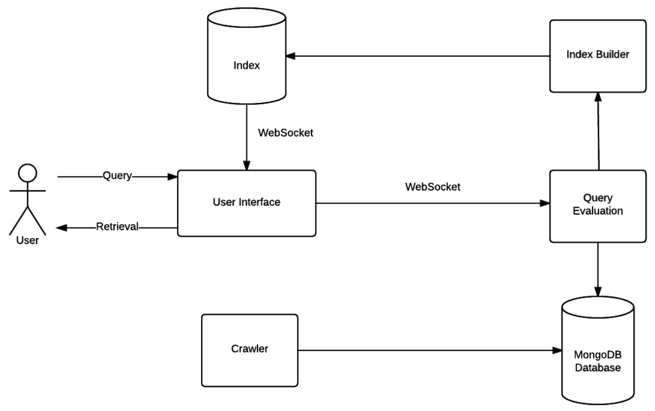
We were asked to build a video search engine to search for videos in a repository like **Youtube**.

System Requirements

The system has to *crawl* Youtube for some specific topic, has to provide a friendly interface to allow the final user to search and browse through the collection of videos.
In particular the system, needs to provide *Pseudo Relevance Feedback* in the retrieval phase.

Main Structure

System Architecture



First Approach

The first approach to the problem was to crawl Youtube with **TubeKit**, a tool for creating one's own crawler. Unfortunately, after going a bit deeper in using it we found out that due to Google changes in its API, the tool didn't worked anymore; we tried to forked the project and modify the code in order to met the brand new Youtube APIs but it was beginning to be painful (mostly because both request and response to and from Youtube did change), and at that point we decided that was the right time to begin to build our own crawler.

Crawler

We implemented a simple crawler in **Python** that requests videos to the API and returns us a collection of results in JSON format, which then is saved in a

MongoDB database. We have decided two main topics to crawl for, **NBA and NFL**. From now on, requesting videos to the Youtube API is possible only by registering an application that gives back a KEY to insert in the API request call.

Crawler Features

The crawler needs to store the following:

- the video URL
- the video title
- the video description
- the video thumbnail

In order to allow the final user to search and browse through the videos, we have chosen two fields to match the initial user query, **title** and **description**. Here's an example of a query to the API for a particular topic:

```
https://www.googleapis.com/youtube/v3/search?part=snippet
&q=<TOPIC>&key=<API_KEY>&maxResults=<NUM_MAX_RESULTS>
```

Youtube API Constraints

- Need for an auth KEY
- The maximum number of results per request is 50
- The maximum number of crawable results is limited by the API to 1000

Due to the limited results (1000) for a specific topic we decided to expand in some way our topic by adding to the crawl process other sub-topics. Trivial correlated videos that we used to expand our crawl are NBA teams, (Chicago Bulls ..etc) and NFL teams (Patriots ... etc).

Note you can find the topic and subtopic list in our config.json file in youtube-crawler folder

Following is an example of response:

```
{
 "kind": "youtube#searchListResponse",
 "etag": "\"3WIcRE7IJ70nCYemJJIi1L7dYAg/7VKCbhm22Y6Hpvj1NWSjrYlKFoI\"",
 "nextPageToken": "CAIQAA",
 "prevPageToken": "CAEQAQ",
 "pageInfo": {
  "totalResults": 1000000,
  "resultsPerPage": 1
 },
 "items": [
  {
   "kind": "youtube#searchResult",
   "etag": "\"3WIcRE7IJ70nCYemJJIi1L7dYAg/hi-7VW72t6lj8VjZxTdMa_Gp0Xw\"",
   "id": {
    "kind": "youtube#video",
    "videoId": "5BFeInvc4OY"
   },
   "snippet": {
    "publishedAt": "2015-12-13T07:46:36.000Z",
    "channelId": "UCWJ2lWNubArHWmf3FIHbfcQ",
    "title": "Top 10 NBA Plays: December 12th",
    "description": "Get ready for Saturday's top ten plays of the night. About the NBA: The NBA is the premier professional basketball league in the United States and Canada.",
    "thumbnails": {
     "default": {
      "url": "https://i.ytimg.com/vi/5BFeInvc4OY/default.jpg"
     },
     "medium": {
      "url": "https://i.ytimg.com/vi/5BFeInvc4OY/mqdefault.jpg"
     },
     "high": {
      "url": "https://i.ytimg.com/vi/5BFeInvc4OY/hqdefault.jpg"
     }
    },
    "channelTitle": "NBA",
    "liveBroadcastContent": "none"
   }
  }]}
```

The most important part of the response are the snippet part, in which are located our fields of interest and the `nextPageToken` key, which provides a string to reuse in the new query to crawl for the next result.

You can find more detailed explanation in the code, all methods are well commented.

Indexing

For the indexing phase, we just simply loaded the collection of crawled videos from the MongoDB database, we use the **Lucene Standard Analyzer** which remove stop words and stem the title and the description of the video, then we add the results to the index together to the other database fields.

Query Evaluation

For the query evaluation part we were asked to build a *Pseudo Relevance Feedback*, so when a user query comes in, we first apply the stopping and stemming as we did in the indexing part, then we collect the top 10 documents that match the query in the title (with a boost of $2.0f$), the body, and the topic.So the final query is a Boolean query built with the 3 query above (title, description, topic) and is an OR query. To the top 10 documents that are considered to be relevant (by the pseudo relevance) we apply the heuristic and we re-enter the query to the system and present the results to the user.

Evaluation and Test

Another core issue of the project was to evaluate and run tests, and so we did. At first glance when a decent version of the system was ready we asked 3 different couples of our classmates to use the search engine.
Thanks to this important phase we learned that user experience takes a very fundamental place in defining the quality of a system, as majority of our *testers* were complaining on the fact, (despite the good response of our search engine), that we were missing a big detail, the videos thumbnail, and surely for a system whose content is very eye-based this was a must. On the other hand as already said the system was pretty well done so for issues like response time and easiness we had no complains.

Technologies

- Crawler entirely written in Python
- Indexer entirely written in Java, Lucene, WebSocket

- User Interface, Javascript, HTML5, CSS3, WebSocket, Node.js, Jade
- Crawled video Database, MongoDB