Information Retrieval - Submission Report
Project N.14: Art For Sale

Jeferson Morales Mariciano, Filippo Piloni
Università della Svizzera italiana, Faculty of Informatics, Lugano, Switzerland

Contents

1 Introduction

The Information Retrieval course project *"ArtForSale"* aims to be a working prototype of an information retrieval system for a specific task: to display for-sale artworks from art selling related websites listed in Section 1.1. The system has around $\approx 8,500$ indexed documents for searching, browsing and presented through a user-friendly and intuitive interface, further details in UI/UX Section 5.2.1, 6.1.1. The source code repository of the project is linked in Section 9.

1.1 Websites

The following websites were chosen for the project based on availability of data, ease of scraping and low restrictions regarding policies and terms of use to scrape documents i.e. *robots.txt* rules:

- `www.artsy.net`
- `www.artfinder.com`
- `www.saatchiart.com`

The cumulative number of documents gathered from the above websites is around $\approx 8,500$ stored in *results.json* file containing the list of samples in JSON format. Further information concerning the scraping process and implementation is discussed in Crawler Section 3.

1.2 Features

The course project requires 2 additional categorized features to be implemented besides the retrieval system: 1 each from simple and complex group features.
The *Art For Sale* project implements 2 + 1 bonus additional simple features:

- Simple
  - **Result presentation**: results should be presented in a tabular format so that many results could be seen at the same time. Each table cell should contain appropriate information for your project.
  - **Filtering**: in addition to being able to search by title, an user should be able to filter the results based on at least 3 relevant attributes for your project.
- Complex
  - **Automatic recommendation**: In addition to the relevant search results pertaining to the user query, the user should also be suggested "similar" products based on, say, category, description, price etc. The ordering among the recommended items is not important. However, you should mention how did you arrive at the recommendations. For this purpose, you can use any open-source recommenders available.

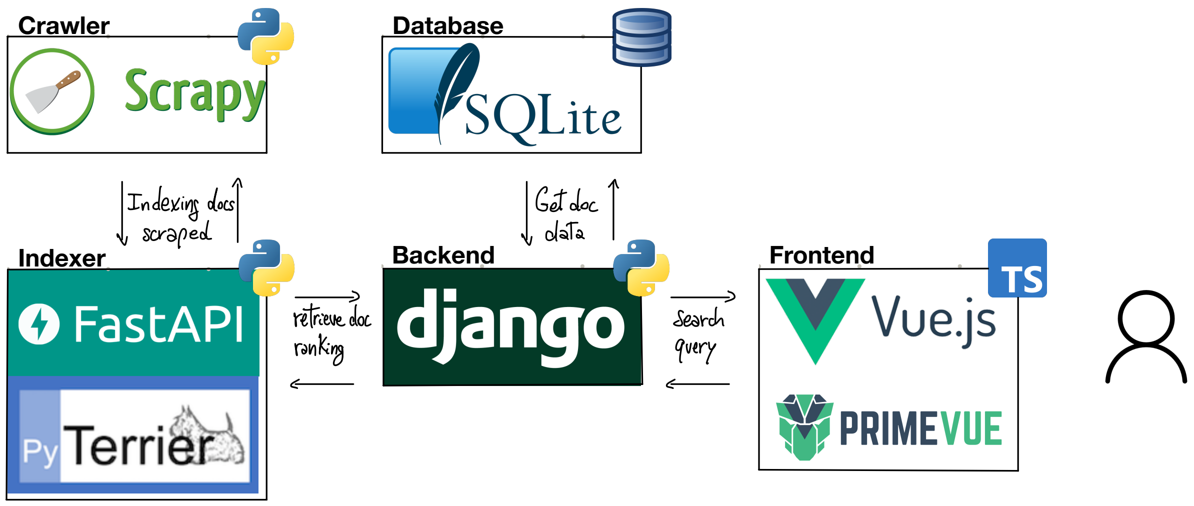
Shown results are sorted in descending order of relevance from left to right in the tabular view. Implementation details are on Architecture, Recommender and Filtering Sections 2, 5.1.1, 5.2.2. Moreover, by exploiting the automatic recommandation feature implemented using category tags and the filtering feature to filter them, the **Results clustering** complex feature was tackled by allowing to group the results into topics based on category tags. Though, a negative filter to exclude documents based on categories was not added, thus partially implementing another extra feature.

2 Architecture

The **Information Retrieval System** has capabilities limited to a **TIS**[1] with **Text retrieval**.
The **Text Access** part helping users to retrieve the right information at the right time, is structured by connecting the system with users in both modes:

- **Pull**: user takes initiative to fetch ad-hoc relevant information from the system, which offers **querying** and **browsing** capabilities. It satisfies **short-term needs**.
- **Push**: system takes initiative and has good knowledge of what are the user's **long-term needs** of information throughout the filtering and recommendation system using **browsing traces** to model the behavior of the user.

Fig 1 Search Engine Architecture of *ArtForSale* project



The software components are: crawler, indexer, database, backend, frontend.
RESTful APIs are used as interface for commucation between indexer-backend and backend-frontend. Communication between database-backend is encapsulated by Django, while crawler-indexer needs to be run manually through CLI prompt and file writing.

2.1 Technology Stack

Mandatory technologies to use for the project implementation: scrapy, pyterrier. Here below is listed the tech stack used with a short description and motivation of choice.

2.1.1 Scrapy

Scrapy is a Python application framework for crawling web sites and extracting structured data. It is used to initialize the crawling & scraping process for a specified set of artworks. Crawling process is further discussed in section 3.

[1]Text Information System

2.1.2 PyTerrier

PyTerrier is a Python declarative platform for information retrieval experiments. It uses the Java-based Terrier information retrieval platform internally to support indexing and retrieval operations. Indexer process is further discussed in section 4.

2.1.3 Django & SQLite

Django is a FOSS[2] high-level Python web framework for rapid development and clean, pragmatic design. It takes care of much of the hassle of web development.
By default, the Django configuration uses SQLite as database, which is the easiest choice of implementation already is included in Python. Backend functionality and interaction with database is detailed in Sections 5.1.

2.1.4 Vue.js 3 & PrimeVue

The progressive javascript framework described as approachable, performant and versatile for building web user interfaces. It is an industry standard among the js component-based frameworks and it is used in courses since the 2nd year of the bachelor.
Together with PrimeVue, a complete UI suite for Vue.js, it elevates the web application with customizable, feature-rich UI components. Frontend design specifics are in Sections 5.2, 5.2.1, 6.1.1.

The search engine requirements are futher described, measured and discussed in the Evaluation Section 6.

3 Crawling

It is part of **Indexing Process**, regarding **Text Acquisition**, in which the system identifies and stores documents for indexing.
The purpose is to harness (relatively) bit text data characterized with the 5 Vs[3] located in previous mentioned web sites regarding for sale art. Among the 5 Vs, automating the Velocity attribute to keep track of the dynamic real-time data characterizing the art selling market was not possible: manual re-crawling of the spiders and inverted index re-creation is needed. Crawling through Scrapy parses text that is unstructed to group it in well-structured documents, allowing them to be easily indexed. The structure of each documents is:

- author
- title
- price
- description
- categories
- image url reference
- post url reference

[2]Free and Open Source
[3]Volume, Variety, Velocity, Value, Veracity

The extracted data is subsequently channeled into a *.json* file specified at start. After all spiders have been run, every result is manually merged into the *results.json* file.
Our sample volume is around $\approx 8,500$ entries. To run crawling refer to project's *README.md*.

4 Indexing

It is part of **Indexing Process**, regarding **Text Transformation**, in which the system transforms documents into index terms, **Index Creation**, where index terms are employed to create data structures to support fast searching, and **Ranking and Retrieval**, which uses query and index to generate a ranked list of docs.
The Pyterrier platform supports indexing and retrieval operations by employing shallow NLP[4] techniques.
The indexer served through a FastAPI server in *crawler/index.py* file, returns a ranked list of documents in descending order of relevance probability $p \in [0, 1]$, which is the optimal standard strategy justified by the **Probability Ranking Principle** [Robertson 77]. The indexer creates the inverted index from the *results.json* file containing the scraped documents. Characteristics of index:

- Stemmer: Porter, algorithmic-basedd, Pyterrier's default
- Stopword: Pyterrier's default
- Tokenisation: English words, Pyterrier's default

The retrieval model is BM25: similarity-based model with ranked retrieval including TF[5], IDF[6] and sublinear TF transformation to avoid single term dominance. To run the indexer refer to project's *README.md*.

5 Retrieval

5.1 Backend

The backend relies on a SQLite database to store the main information about the documents, making it possible to retrieve the relevant ones following a call to the index.
The backend handles the two main fetches of the application:

- *GET /api/documents/get-documents/$\underline{query}$/*: Endpoint facilitating the retrieval of relevant documents based on a provided $\underline{\text{query}}$ string.
- *GET /api/recommendation/get-recommended/*: Endpoint tasked with retrieving recommended artworks. By inputting a set of tags, it conducts a search within the index, returning all pertinent documents associated with the specified category tags.

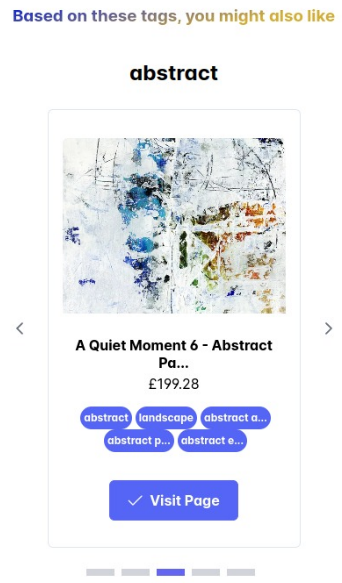
The backend is the joint between the frontend and the indexer. It routes fetch request and, given a query, it returns the list of relevant documents with its main information by iterating through the database containing it for each artwork.

[4]Natural Language Processing
[5]Term Frequency
[6]Inverse Document Frequency

5.1.1 *Automatic recommendation*

The 2nd API call is used to implement the *recommender system*: the frontend keeps track of the categories as **browser traces** from the results given by user's queries. Then, the indexer receives such categories as query and give back a set of documents to the backend. This set is populated with its information stored in the database and it is finally given back to the frontend in the suggestions carousels. Essentially, once all the artworks related to the query have been retrieved, the frontend extracts from all these artworks the related tags, it chooses the ones with the higher frequency, fetches result using the API call and shows a maximum of 5 results related to each tag in a moving carousel It is a **Content-Based Information Filtering**: recommendation in such dynamic collection is individual to users based on interest and preferences inferred from previous user pseudo feedback, hence assuming top retrieved documents are relevant. At the end of each section the recommendations tags are cleaned i.e. page reload.

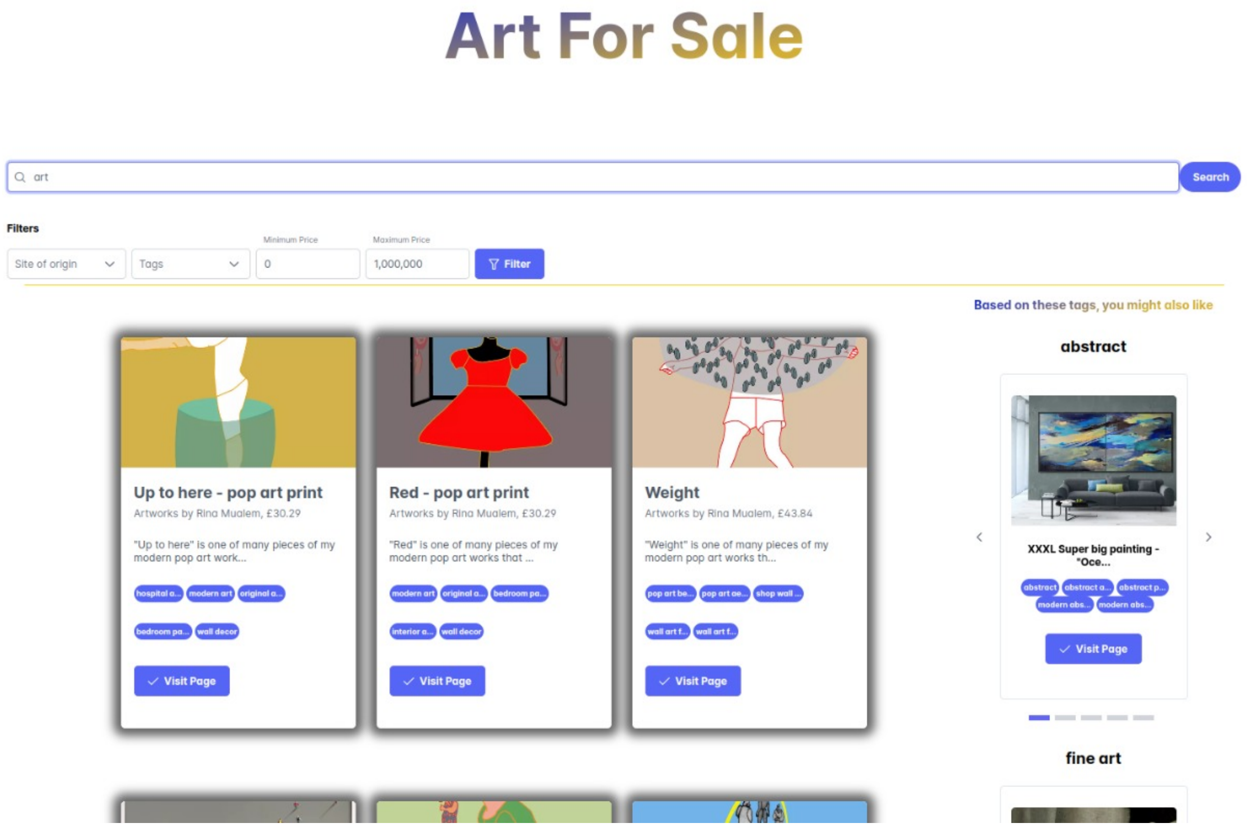
Fig 2 Automatic Recommendation - Carousel



5.2 *Frontend*

Part of **Query and Retrieval Process**, regarding **User Interaction**, in which the system supports the creation of a query and display results.
The frontend is the main part of the application, where all the features are implemented. For the interface, it has been chosen a simple layout shown in Figure 3, consisting of only a search bar and a search button. Once the user performs the search, the aspect of the page changes, adding new components:

Fig 3 UI of the *Art For Sale* project



Up to here - pop art print
Artworks by Rina Mualem, £30.29
"Up to here" is one of many pieces of my modern pop art work...
hospital a... modern art original a... bedroom pa... wall decor

Red - pop art print
Artworks by Rina Mualem, £30.29
"Red" is one of many pieces of my modern pop art works that ...
modern art original a... bedroom pa... interior a... wall decor

Weight
Artworks by Rina Mualem, £43.84
"Weight" is one of many pieces of my modern pop art works th...
pop art be... pop art ae... shop wall ... wall art f... wall art f...

XXXL Super big painting - "Oce...
abstract abstract a... abstract p... modern abs... modern abs...

- Filter selectors: under the search bar there are filter selectors to control the obtained results according to 4 main characteristics: site of origin, minimum price, maximum price and category tag.
- Recommended Carousel: At the right side of the page, the recommended carousel suggests results based on the most frequent tags of the resulted artworks. It shows a maximum of 5 results related to each tag in a moving carousel.
- Document tabular view: retrieved artworks from the query are displayed in card components, following a tabular view as requested by the simple feature implemented. The card shows an insight of artwork image, author, title, price, short description and main tags, with a *"Visit"* button that will bring the user to the page of that artwork in the original website.

5.2.1 User Interface

UI has been layed out following Jakob's 10 usability heuristics as much as possible, particularly: aesthetic and minimalist design, visibility of system status and consistency and standards.
Norman's Emotional Design guidelines have been followed to obtain a UI gradient effect and minimalism similar to what is the user experience in Apple products, so users will feel confortable using it.

5.2.2 Filtering

The filtering features is implemented frontend side. If no filter is selected, then it simply shows all the results from the query. Otherwise, it checks:

- price value range: check if the document value is between the min and max inserted
- web sites: check if the document origin url contains the selected website(s)
- category: the UI allows only to select a category from the ones already present in the result of your query

6 Evaluation

Part of the **Query and Retrieval Process**, regarding **Evaluation**.

Employing of **query expansion** was tested by always adding the *art* keywork at the end of each search to get more results. This led to under-constrained query giving back not relevant results. The reason was due to the fact that for our specific set of concerns, hence documents, the word *"art"* is an high frequency word, thus useless to achieve pertinency as explained by **Zipf's Law**.

At start, every field was in a *pandas.Series* and then given to the indexer but this lead to unsatisfying results: it seemed like only the first field (title) got indexed, giving an **over-constrained query** feeling where either no documents or no relevant ones were returned, Hence we decided to put every field into one big "blob" of text and then indexed that.

Effectiveness: quality of results. **Efficiency**: response time and throughput.

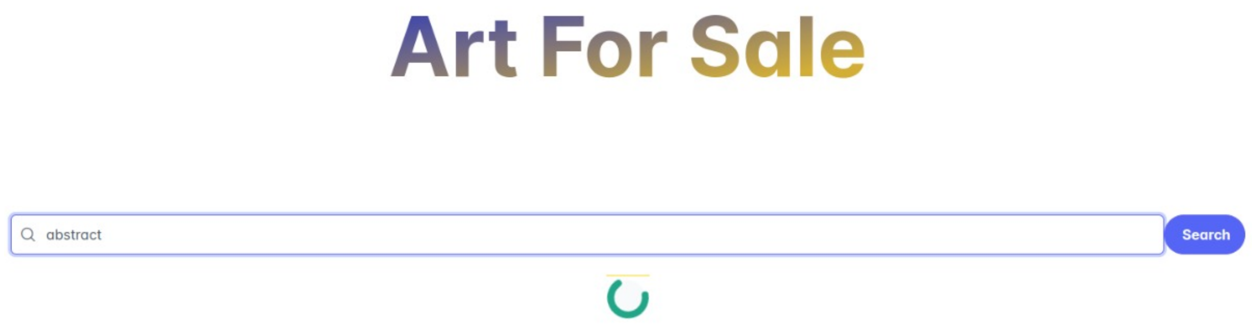
System evaluation: Recall and precision paragraph

6.1 User Evaluation

Part of the **Relevance Feedback Process**, regarding **User Evaluation**.

SUS includes someone normal not studying informatics, quote are designers users too?

Fig 4 Loading animation while retrieving documents



6.1.1 User Experience

Sentiment analysis

7 Limitations and Future Improvements

The project is a *Text Retrieval* system at its current state. Since the type of data provided by artworks is not descriptive nor verbose, implementing *Text Analysis* features becomes troublesome because lacking the large amounts of text data in order to discover interesting patterns buried in text. Nevertheless, *Text Organization* implementation feasibility is high due to short well-structured categorized-based descriptions usually artwork documents have, allowing to connect and navigate scattered information.

8 Conclusions

We had fun with deciding the technology stack to use and the architecture structure. We enjoyed the project freedom that allowed us to put in practice all the methologies learnt from this course and all the others we had during the bachelor.

9 References

- *ArtForSale* GitHub public repository: `https://github.com/JekxDevil/IR-ArtForSale`
- Course slides on iCorsi: `https://www.icorsi.ch/course/view.php?id=16928`
- *Scrapy*: `https://docs.scrapy.org`
- *Pyterrier*: `https://pyterrier.readthedocs.io`
- *FastAPI*: `https://fastapi.tiangolo.com/`
- *Django*: `https://www.djangoproject.com/`
- *Vue.js 3*: `https://vuejs.org/`
- *PrimeVue*: `https://primevue.org/`