Project: Tracking the flow of historical artifacts using open data sources

Team members: Nazariy Perepichka, Volodymyr Lut, Roman Riazantsev, Yarema Sirskiy

Link to the repository: https://github.com/baaraban/sparkyShinyIndianaJones

Introduction

This is the report for the final project for “Mining Massive Databases” course.

Our team decided to build a tool to visualize and track transitions of historical objects interactively.

We gathered open-source data, examined and applied various machine learning techniques, processed the data with pySpark, and visualized it with RShiny.

Data collection

We collected the data from two open sources.

First, we scraped information about the Louvre exhibits from its official site. The museum's staff organized collections of their showpieces in a well-structured and productive manner: each page describes artifact with an article and info tables(name, category, inception year, etc.). Additionally, we definitely know the current location of these artifacts.

Also, we created the tool for extracting information from Wikidata. The script crawled through pre-defined parent categories and formed the dataset with a similar structure to Louvre’s one.

The second dataset forms the core of all gathered data; the first one was mainly used for development and testing.

Modeling

After we retrieved the data, we started building the NLP model to extract locations of objects through time. This task is non-trivial, and we haven’t found existing solutions for the problem.

We decided to build a custom algorithm for such a task:

1. Parse text and locate sentences with mentions of artifact
2. Recognize named entities(locations and dates)
3. Choose the best location and date pair

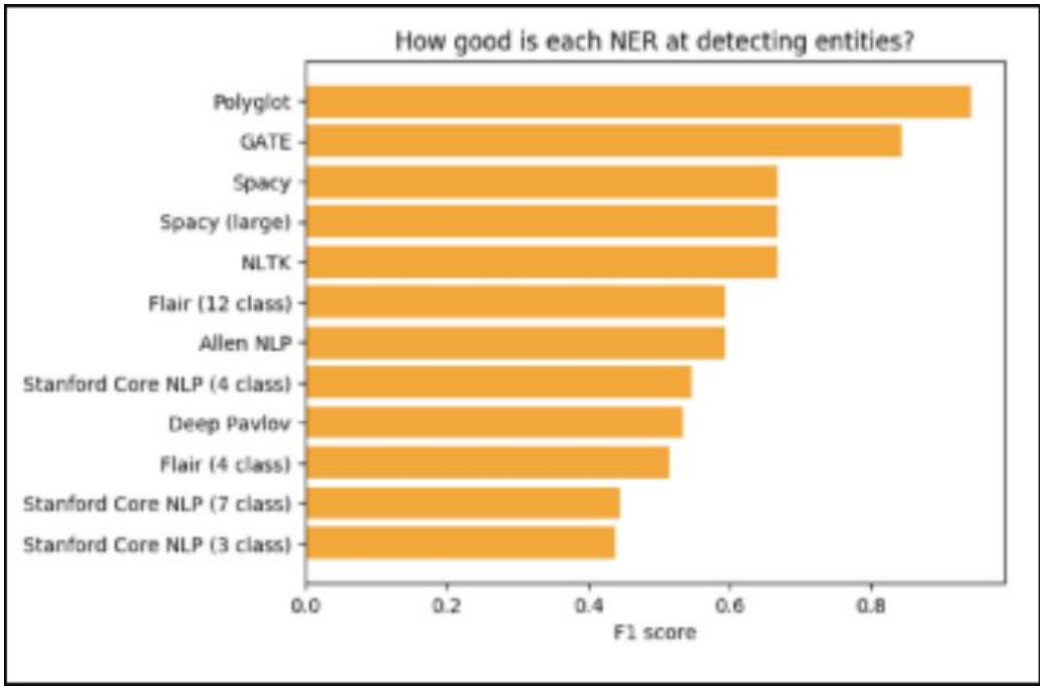
The first part of our algorithm is known as entity linking. We didn’t find a lot of “ready-to-go” solutions for it. We tried textacy package[1], but it performed poorly and didn’t detect mentions of artifacts. We stubbed the first part of the algorithm by finding all the sentences with the following words: name of the entity, aliases of the entity, literals(statue, tool, portrait, peisage, treasure, blade, painting, ax, jewelry, fresco, arrow, the work).

We proceeded to the second part of the algorithm and researched possible solutions for Named Entity Recognition problem. Because our dataset was unlabeled, we tried pre-trained models available in the open-source community: Spacy models, deeppavlov and polyglot.

Spacy documentation gives the following scores for NER models[2]:

	en_core_web_sm	en_core_web_md	en_core_web_lg
F-score	85.86	86.56	86.62
Precision	86.33	86.88	87.03
Recall	85.39	86.25	86.20

The other two packages do not have scores in the documentation, but during the research, we read the blog post that compared different NER solutions[3]. It gives the following comparison diagram.



We tested the models on the small subset of our data - one hundred sentences from different texts - and retrieved the following results:

	en_core_web_sm	en_core_web_md	en_core_web_lg	deeppavlov	polyglot
F-score	75.34	78.43	62.41	63.36	49.23

The current solution uses ‘en_core_web_md’ Spacy model because of the highest score.
The model gives location in three possible forms:
‘GPE’ - Countries, cities, states;
‘FAC’ - Buildings, airports, highways, bridges, etc.
‘ORG’ - Companies, agencies, institutions, etc.

For the third part of the algorithm, we prioritized those tags, in the order, they are described above.

For each sentence we formed all of the possible date-location pairs and scored them using the custom formula:

Score = (1 / dd) + (1/(priority*ld)), where:

dd - distance from date to mention of an entity
priority - priority of location tag
ld - distance from the location to mention of an entity

Processing

The next step was to process all the collected data using our algorithm. We used pySpark for this task.

We encountered issues with spacy model serialization. To resolve them, we lazy-loaded spacy models, ensured the existing of only one instance per context and used pandas_udfs for data manipulations.

Each data source differs in the structure, so for each of them, we implemented the source processor class that encapsulates transformation logic from raw data frame to the unified format. Our solution has the processing manager class, which handles all of the source processors, Spark context, data loading, and saving.

Visualization

After processing of the datasets, we received dataset in the following format:

artifact_name	year	lat	lon	image_link
Name of the artifact; serves as id	Year in which artifact was in a specific location	Latitude of location	Longitude of location	Link to image of the artifact

Based on this data, we build the visualization[4]:

Currently, the tool displays transitions of objects and allows to filtrate artifacts by years. That's only part of planned visualization - you can see it from the data included in the final dataset - but that is what we managed to do before the deadline.

Possible improvements

- **Data Collection**

The most reasonable thing to do is to switch from saving data in CSV files to storing the data in the database. It can help to speed up the process of processing the data and organize data better.

- **Modeling**

We should create a custom model for our problem: train the LSTM-based model for extracting date and location-specific to the artifact.

The main challenge is to find or create a labeled dataset for training.

- **Processing**

Right now, we don't understand the progress of the processing step. We should implement a batched approach and add a logging system.

The system is not robust to errors and should be tested more.

Because it was the first experience with Spark for every team member, we suspect that the current code can also be optimized and structured better.

- **Visualization**

Our current map requires more style. We plan to add arrows for the direction of transitions, add more filtering options, and create tooltips for each artifact.

The performance of the R-based leaflet map is not optimal too. We should discover other possibilities, which allow plotting massive arrays of data-points efficiently.

- **Ways to extend the project**

We can scale the project in "depth and breadth."

Adding more data sources, types of artifacts and visualization options will make our project more complete.

We can also add extra layers: analyze movements of artifacts, cluster the data, detect anomalies.

Summary

We created a fully-functional solution that satisfies initial requirements. Though all of the layers need improvements, we think that our project is a practical MVP and a starting point for future this project.

We finish this report with eager to continue the development and gratitude to the reader for reading it to this point

References

1. https://pypi.org/project/textacy/
2. https://spacy.io/api/annotation
3. https://medium.com/@b.terryjack/nlp-pretrained-named-entity-recognition-7caa5cd28d7b
4. https://volodymyrlut.shinyapps.io/artifacts/