JavaScript Technology Seminar 2017 Group 2: Unstructured Data - Report

***Felix Schorer, Lukas Streit, Nikita Basargin, Anshul Sharma**, Technical University of Munich*

The goal of the `MusicConnectionMachine` project is to create a “social” network for classical composers, music pieces and musicians. There are a number of different relationships between them: piece A was written by composer B, musicians C and D both liked genre E, composers F and G lived in the city H at the year I, etc. Information is extracted from different internet sources. After processing and classification, the results are visualized and made open to public by integration into a popular online resource.

Sub-projects and teams

The seminar initially had three main sub-projects: *data aggregation*, *information extraction* and *visualization*. Two independent teams with four members each were assigned to each sub-project. During the seminar some structural changes were made. Some members actively worked on the interfaces and the new *API* sub-project. Some teams were merged.

Unstructured data

Our team was responsible for data aggregation on unstructured sources. Data from *CommonCrawl* (CC) was used. This dataset is hosted on Amazon S3 and stores web page data from the last 7 years in different representations. We decided to work with text extractions (WET files) from the latest crawl. The biggest challenge was the size of the data. The March 2017 crawl produced 66500 WET files and more than 9 TB text extractions (compressed). Processing this amount of data on one machine would be too slow. Therefore we used multiple VMs hosted on Microsoft Azure for parallel processing.

A lot of data from CC is useless for us because it contains no information about classical music. This data must be quickly filtered and should not be passed to the groups dealing with information extraction. Otherwise it would significantly slow down the processing since natural language processing algorithms are quite slow. The group working on structured data sources provides us with a list of relevant terms. These terms are used for the filtering.

Data processing pipeline

This section explains the basics of the data processing pipeline. In order to keep is simple, we assume that only one machine is doing the processing. Parallelization is described in the next section.

Apart from the command line arguments and environment variables we have two main inputs: terms provided by the structured data group and the actual data from CC. The terms are stored in the database. After loading, we remove those terms that are present on our blacklist. Afterwards we download WET files from CC one at a time, unpack them and start filtering.

We tried multiple filtering approaches such as naive string search, bloom filter, prefix tree and some combinations of those. Currently we use a prefix tree constructed from the relevant terms. For each web page in the WET file, we check how often relevant terms occur in the text. We use a heuristics: The web page is considered relevant only if n distinct terms are found and there are at least n^2 total term occurrences.

As an additional filtering step we use language detection. Only web pages in English are passed. If a web page passes the filtering step, the whole content is stored inside a blob storage on Azure. The metadata (link to the blob, what and how many terms were found) are saved inside the database.

Parallel processing

Each WET file is independent from others. This allows very high parallelism. On a single VM we have one master process and m worker processes. The master loads terms from the DB, spawns the workers and assigns tasks to them. Each worker processes a single WET file at a time and informs the master once it is finished.

During the last seminar week shell scripts to spawn multiple VMs were added. A global queue is used to pass messages to all VMs.

Internal task distribution

All team members worked on their parts of the presentation. Detailed code statistics are available in the git repository.

Felix Schorer mainly worked on filters, downloader, web page digester, master-worker processes, CLI. He also helped to define the swagger schema in the API sub-project. Most tests and refactoring were done by him.

Lukas Streit mainly worked on storer, WET manager, term loader, database interaction, deployment scripts, blob storage and Azure infrastructure. He also worked on the API sub-project.

Nikita Basargin mainly worked on filters, unpacker, CC-index, CLI, first test runs and provided sample data to information extraction groups. He wrote this report and attended all meetings except the last week's hackathon.

Anshul Sharma mainly worked on language extractor, docker file, TravisCI and some other minor issues.

Timeline

After the inspiring kick-off event we started to inform ourselves how to get access to the CC data. Some basic functionality like downloading, unpacking and WET file parsing was implemented.

In the middle of the project we had to switch to another topic: RESTful APIs. Presentation and the workshop preparation took some time. Introduction to Azure after the presentations was very helpful.

After the presentations the main implementation phase started. Some ideas like TF-IDF had to be dropped due the large amount of data.

Scaling to mutiple VMs was implemented during the final week. Also, continuous improvements and refactoring took place.

Problems and feedback

One of the problems was the late introduction of the API sub-project. For a very long time interfaces were not defined and that slowed down some groups. For the next project it might be helpful to have a group entirely responsible for interfaces and communication only.

At the beginning of the seminar there was a lot of competition not only between groups but also inside of those. Competition sometimes can be motivating. However, it also can have the opposite effect if a group or member can't keep up with others.

Team meetings showed to be productive. For the next seminar we could have more of them.

Conclusion

The seminar was quite challenging and required a lot of new things to be learned. Fortunately, a lot of work has led to an impressive result. We are able to process huge amounts of data from CommonCrawl using the computational power of many VMs. There are still a few things that can be improved and I'm happy to see that some team members continue to work on them even after the end of the seminar. A special thanks to all JST mentors who supported us during the seminar and made everything possible.