Beeg Data - Beeg Meme

Kacper Grzymkowski, Jakub Fołtyn, Mikołaj Malec, Illia Tesliuk

Introduction

This document represents the *Beeg Meme* project state at the stage of the 3rd milestone delivery in the *Big Data Analytics* course. In it, we will describe the final shape of our data preprocessing step, as well as the Exploratory Data Analysis that we have conducted and all the details about the *Analytical module* of our project. We will also provide a renewed system architecture diagram and plans for the final steps of our project. Finally, we will also describe the work division among team members during this milestone.

Changes to the 2nd document

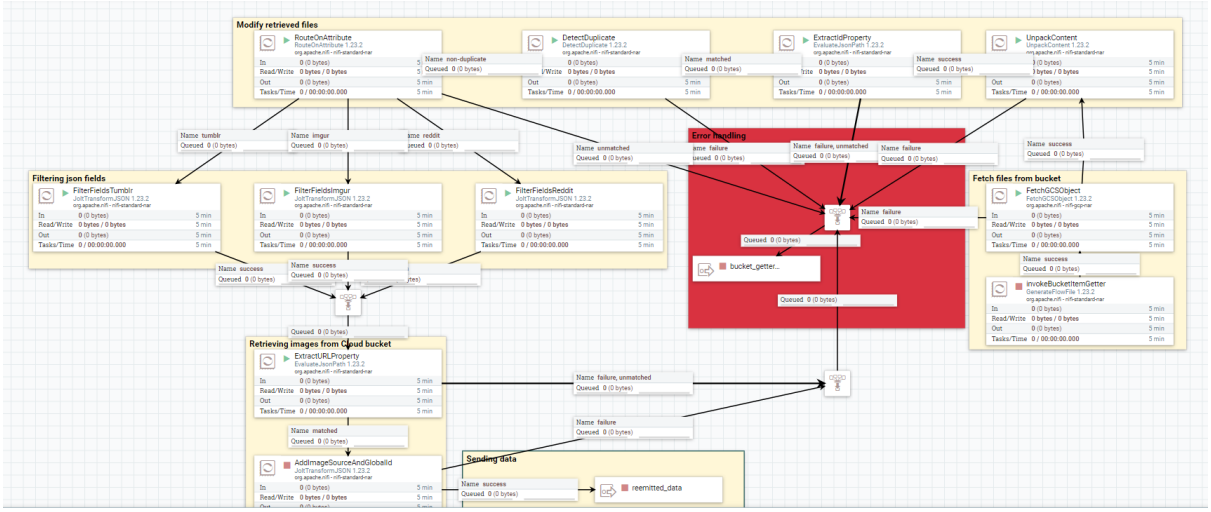
- The Nifi data ingestion module has been upgraded – details in the “Data preprocessing” section.

Data preprocessing

For the purpose of our analytical module, only a small number of fields from the source JSON files are used. As such, the preliminary preprocessing done in the Nifi module has stayed the same. We are still unifying the fields among all three sources (Reddit, Imgur, Tumblr) and posting those, unchanged, to the Kafka module. As a reminded, the fields that we are filtering are:

- **author** – author of the post.
- **desc** – description
- **createdate** – UTC date of image creation
- **score** – post “score” (number of upvotes-downvotes)
- **source** – image source
- **id** – image global id, created by merging local image id with its source
- **url** – url address of the image

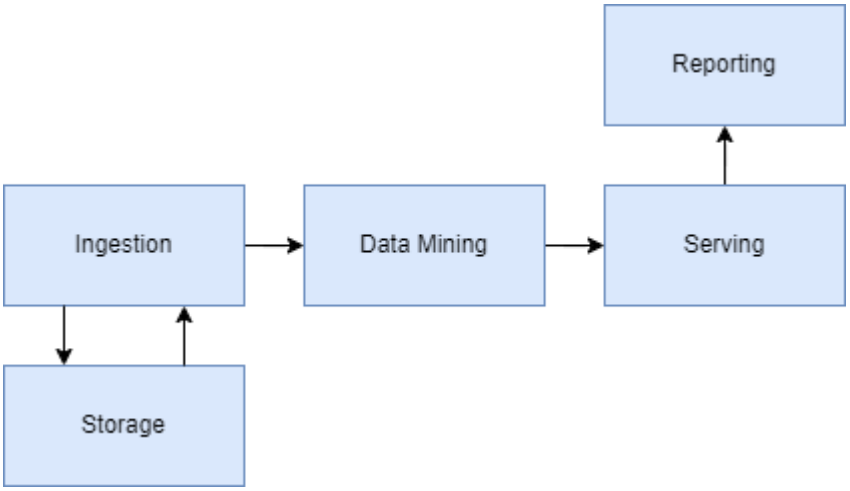
There have been, however, some changes made to the Nifi module at large. Importantly, we have added two additional scripts: *loadImage.py* and *saveToBucket.py*. By using them, we are now able to extract the image when first obtaining the post from the source and then save that image into a dedicated *Google Cloud Storage bucket*. This bucket is named *images_beeg_meme*. Each image is stored under its original URL. That way, when we retrieve some images from the past, we are certain that even if the original URLs do not work anymore, we can still retrieve the image copy from that bucket.



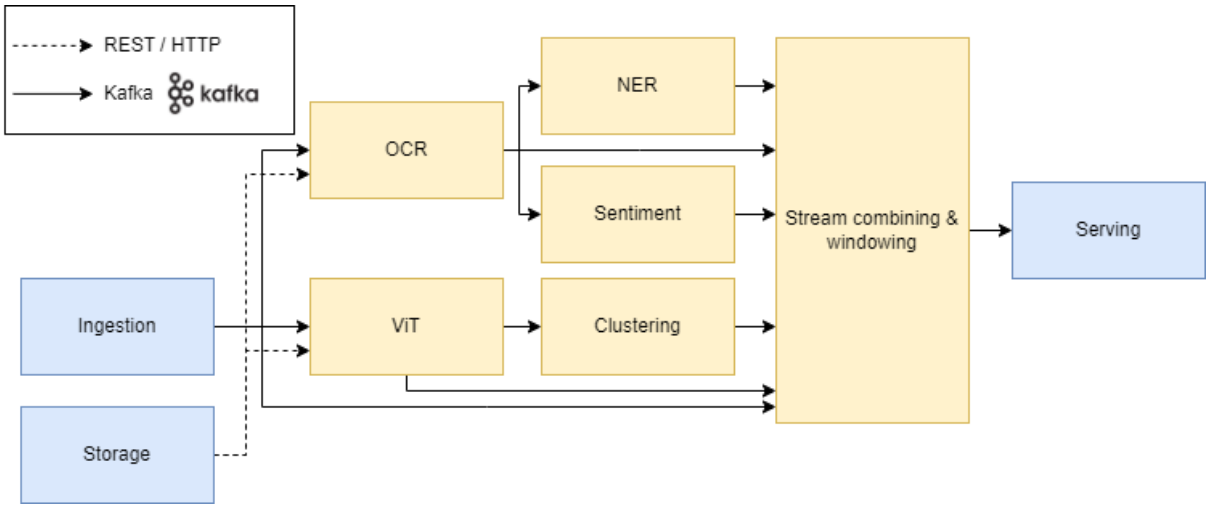
Furthermore, we have implemented the re-emittance path. Now, we can access any of the previously retrieved posts. As a reminder, they are stored in *Master Log GCS bucket* in *.tar* files named after the date and time of their storing. These files usually contain around 10,000 posts. Through the re-emittance path, we can now download any of these files and use posts collected inside as part of it. Those posts are then deduplicated and filtered (just as in the “normal” path). What is more, as we are retrieving posts from potentially a long time back, we are replacing the original URL address in the post with the address of a copy made in our bucket – as we cannot be certain if the URL is still accessible.

Complete system architecture

Overview diagram



Data mining diagram



Analytical module

We run a number of machine learning models in order to mine useful information out of the images we process. In particular, we used a pre-trained MobileViT2 model from HuggingFace - a lightweight version of the ViT transformer designed for mobile phones. This original ViT model is too heavy-weight and slow for our system, therefore a faster transformer was required to achieve acceptable inference times. Currently, MobileViT2 requires approximately 1 second per image with part of the time being taken by image downloading. MobileViT2 is used for image processing and produces more compact image representations which are next used by the clustering algorithm.

Since we are implementing a Kappa architecture, a standard image classification model or even K-Means clustering algorithm are not applicable. With streaming architecture in mind, we've selected an online BIRCH clustering algorithm and used its implementation from scikit-learn library. We've selected BIRCH due to its iterative nature that allows it to process images one by one as they come from the ViT module, and not the whole dataset at once. We've already tested it and received non-error cluster id as outputs for input memes, however it's difficult to evaluate the clustering performance, as it requires a larger amount of collected data to draw conclusions.

On another branch of our system we perform Optical Character Recognition on the input meme images. We've used an EasyOCR python module which provides a reader capable of detecting english words on images. It can be said that the algorithm performs quite well and correctly detects the majority of letters, however, sometimes it can make mistakes in a couple of letters. For example '2020's' was interpreted as '2020$' by EasyOCR. OCR operation showed a very inference time. It reaches around 10 seconds on CPU and is even slower on the cloud. However, OCR operation is crucial for our project as there's no other ways to extract text from memes.

Next, the extracted text is sent to Named Entity Recognition (NER) and Sentiment Analysis modules. We use a WikiNEuRal multilingual pipeline for the task of NER and roBERTa-base model fine-tuned onTweetEval benchmark for the task of Sentiment Analysis. Both these pre-trained models were taken from the HuggingFace. While the choice of OCR implementations is quite limited, HuggingFace offers a wide range of models for different NLP tasks or Visual Transformers, therefore we'll be able to test other pre-trained NER and Sentiment Analysis models as well.

All three HuggingFace models we used, achieved SOTA results on their respective benchmarks. It is difficult to evaluate their performance on our data as we do not possess stationary dataset with NER, OCR or Sentiment Analysis labels. Contrary to these three tasks, clustering is an unsupervised task and possesses a list of evaluation metrics that can be applied to our system. Sklearn offers several evaluation metrics that do not require ground-truth labels, namely silhouette coefficient, homogeneity, completeness, V-measure such, Calinski-Harabasz score, Davies-Bouldin index, contingency or pair confusion matrix. Either of these metrics can be used to evaluate the clustering results as they assess how well the resulting clusters are formed. We will apply some of these metrics when we'll collect enough data to perform a sensible analysis.

Each module is implemented with the help of pyspark Python library. Each of the modules receives Kafka messages from the corresponding sources and sends output Kafka messages to the analytics part and other modules.

Tasks planned to finish the project

- Prepare Kafka/Spark for fan-out scaling where possible
- Implement, test and deploy missing components
- Collect data and prepare a faster replay of the stream
    - The amount of data is less than we expected

Tests

The test document is located here.

Work division

Software and report parts

Jakub Fołtyn:
- Nifi module upgrade
- Nifi testing
- Introduction, preprocessing sections

Kacper Grzymkowski:
- Spark set-up
- Kafka set-up
- ViT module implementation and testing
- Clustering module implementation and testing

Illia Tesliuk:
- OCR module implementation and testing
- NER module implementation and testing
- Semantic Analysis module implementation and testing

Mikołaj Malec
-