ImTrends - The Trends of Images in World Wide Web

Norvig Award - Group 7 - Delft University of Technology

View the result: Image Trends! web-application!

[The following report can also be downloaded here]

Feng Wang

Hong Huang

Vincent Gong

1 Idea

An important goal of data analysis is to acquire knowledge from data. A huge amount of research has been carried out in the past to investigate how to obtain knowledge from text. However, the effectiveness of text-based techniques largely depends on the type of language that the particular piece of text is in. The technique that works well for English does not necessarily proves effective for Japanese. Moreover, the effectiveness of text-based techniques may be jeopardized when the target language is used in an informal way, for instance, when a lot of slang is involved. Since there are lots of languages in the world and they are all different, analyzing the text in the Internet seems difficult to get similar results for all languages.

So we take a different approach-to analyze images, which is kind of a universal format of data. The basic idea is to cluster images in a time period and to get the trends of images in that period in the world. The final outcome should be similar to Google trends, but in image form. We want to use our knowledge and experience of measuring the similarities of images using Matlab/C++ in the context of big data processing.

2 Method

In ImTrends there are two essential parts: the data processing part and the visualization part. In the following we describe how these two parts are designed and implemented. An overview of the workflow is showed in Figure 1.

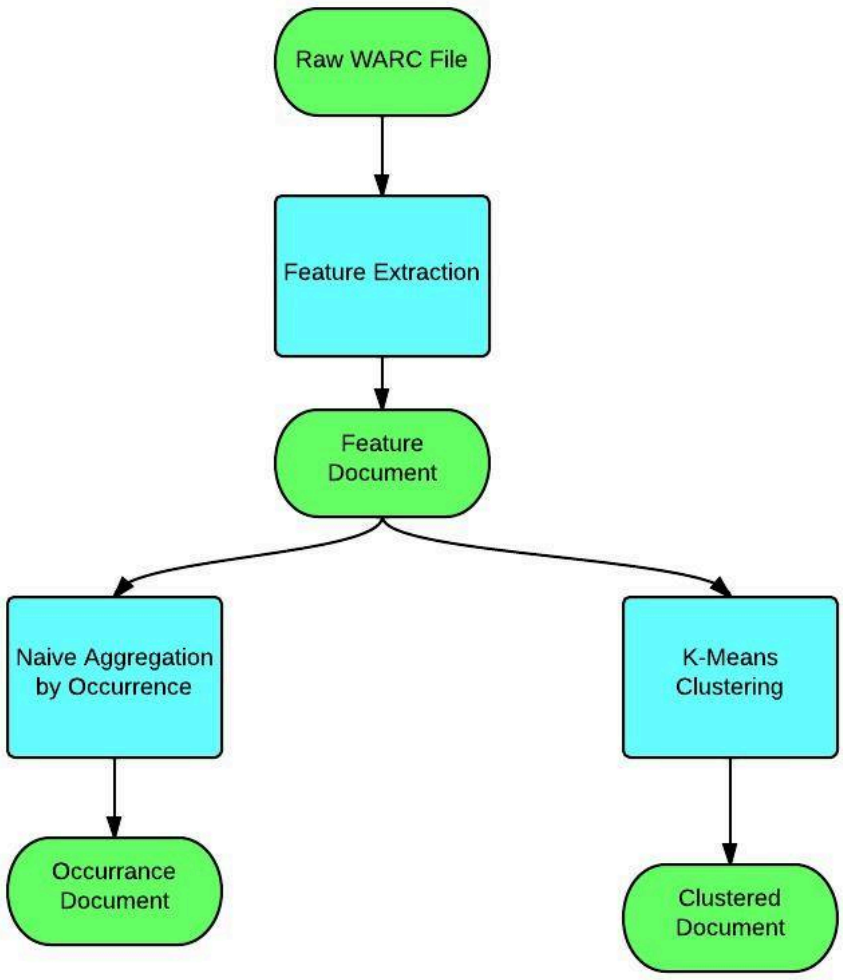

Figure 1: Overview of the ImTrends workflow

2.1 Data Processing

We use the big data from Common Crawl. And we have implemented a map-reduce program in Java for Hadoop. This program runs on SARA's Hadoop cluster and processes the WARC files.

In the data processing part there are two phases:

1. Feature extraction phase: in this phase we extract features from each image in the data set. The output is a file with a list of tuples. One tuple for one images. Each tuple has the feature vector of the image in the form of hash code, the URL for the image and the information of the image (height, width, etc.) and the period the image is in (in our case, the month that the image is in).
2. Aggregation phase: in this phase we use the output from the previous phase as input and find the 'hottest pattern' in each period.

In the following we describe how each phase is implemented.

2.1.1 Feature extraction

In this phase, all we need to do is to apply feature extraction algorithm to each image in the data set. Therefore, from the big data point of view, the feature extraction algorithm works in the mapper phase and no reducer phase is needed in our case.

In sequence, we performs the following steps:

1. For each WARC record, we extract the HTTP header.
2. For the HTTP header, we check the Content-Type field, and discard the header which does not contain "image" in Content-Type fields.
3. Then we check the Last-Modified field in HTTP header, and discard the header without this field. We do that because this is the only way we can know the creation or modification time of the image, and we need to use this time in our clustering part. But an obvious drawback is that lots of images, which do not contain this field in the HTTP header, will be discarded.
4. Check the record payload size. If the size is bigger than a threshold, the record will be discarded. The reason is that some images are extremely large, which is not suitable for the following processing (will result in error in Java ImageIO read function)).
5. Extract the hash codes from the images. We use two methods and extract two hash codes from each image. The details of the extraction methods will be discussed following.

The feature extraction algorithms we use are hash algorithms, which use features of images to generate comparable fingerprints in the form of hash code. Two different perceptual hash algorithms are used here, average hash algorithm and pHash algorithm.

Average Hash Algorithm

This idea comes from LOOKS LIKE IT. And the basic steps are:

1. Crush the image down to fit in the 8x8 square. As a result, there are only 64 pixels in the square.
2. Convert the image to gray scale.
3. Compute the mean value of the 64 pixels.
4. Compute the bits in this way: set to 1 if above the mean, otherwise 0.
5. Finally, turn the 64 bits into a hash code.

pHash Algorithm

The pHash algorithm extends the previous approach by using discrete cosine transform (DCT) to reduce frequencies in images. The image is fit into a 32×32 square instead of 8×8 for the convenience of DCT. When the resulted image is converted to gray scale, use 32×32 discrete cosine transform and keep the top left 8x8 of the result. Based on the result, the mean value is calculated excluding the first discrete cosine term. Finally the bits and the hash code are constructed just as step 4 and 5 in the previous algorithm.

Here we use a Java implementation of pHash for images by Elliot Shepherd.

In the following part, we choose the Average Hash Algorithm.

2.1.2 Aggregation

Essentially, the aggregation is a clustering process. By doing the clustering, some big clusters can be found, which represent the hottest images in this time period. And we have tried different kind of methods.

Naive aggregation by the occurrences of the hash codes

We firstly tried to do the clustering by checking the occurrences of the hash codes. Obviously if a hash code occurs many times, it is popular.

This part is implemented with Pig Latin, which results in a series of Map-Reduce programs. The input of this phase is the output of the previous phase. The Pig script first load the data from the input file, then count the occurrence of each hash code in each period, for instance, January,2010. Then in each period, the hash codes that have the top N occurrence are extracted. This result is joined with the original table to retrieve the detailed information of images that have these hash code pattern. The result of join is then ordered and stored in the output file.

This method can be regarded as a Mean Shift clustering with a width parameter 0. Since the width parameter is 0, this algorithm is extremely quick, but the results seems not so good. Then we also try to use different width parameter, but the efficiency is extremely low ($O(N^2)$). And in each iteration of the Mean Shift clustering, it needs to calculate the distances between all points and the clustering centers, and then sort the distances, so it cannot be done distributedly.

K-means clustering

Considering the problems of Mean Shift clustering, we then tried to use K-means clustering. Since the K-means clustering does not need so many memory in calculating, we directly do it locally by using Python library PyCluster.

For K-means clustering algorithm, the most important parameter is the number of clusters, and we just intuitively set it to 5000, and if the number of data points is smaller than 5000, we set the number of clusters as 0.5×(number of data points).

2.2 Visualization

In this phase a web application based on HTML and PHP is built to visualize the result of the data processing part. The detailed information of the 'hottest images', the images with the most occurrence in a month, are displayed. The detailed information can be queried by time period they are in. For more details, please directly try our website!

3 Result

In this section, we will briefly introduce some interesting results got from Image Trends.

3.1 Statistics

There are in total 10,330,607 images gathered and used in calculating and clustering. The number of images in each month are showed in Figure 2:

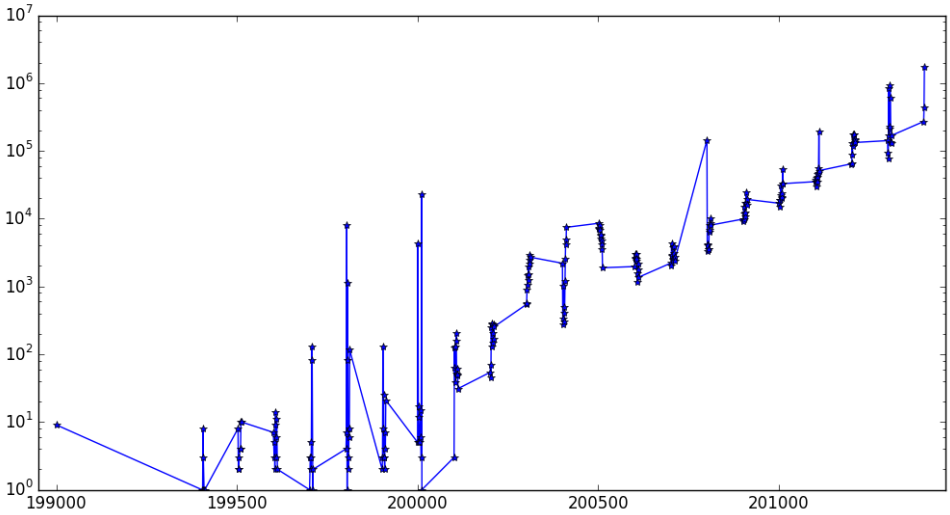

Figure 2: Line chart of the number of images gathered in each month

From this image, we can see that in some months, there are lots of images uploaded or created, and in some neighboring months, not so many images created. This phenomenon is obviously against the common sense that in neighboring months, the number of created images should be similar. We think the reasons for the uncommon numbers could be:

- In some month, lots of images are created by some specific websites. For example, all the images in November 2000 are created by http://cdnbakmi.kaltura.com. Noticed here the "created" means the Last-Modified field we extracted from the HTTP header, and does not mean the real creation time (which we cannot know).
- In the month when Common Crawl do the crawling, more images can be obtained. One example is in March 2014, there are 1,716,437 images, and in February 2014, there are only 438,319 images. And the last time Common Crawl do the crawling is March 2014. This explanation is reasonable since lots of websites tend to delete the useless images a few months after they are created.

3.2 Clustering Analysis

Does the clustering algorithm really work? We answer this question by visualizing the results. Each hash code, extracted using Average Hash Algorithm], is a 64 dimensions binary code. We firstly use PCA to reduce the dimensions to 2. And then use Python matplotlib to draw the scatter plots to show the clustering results. Here are some examples:

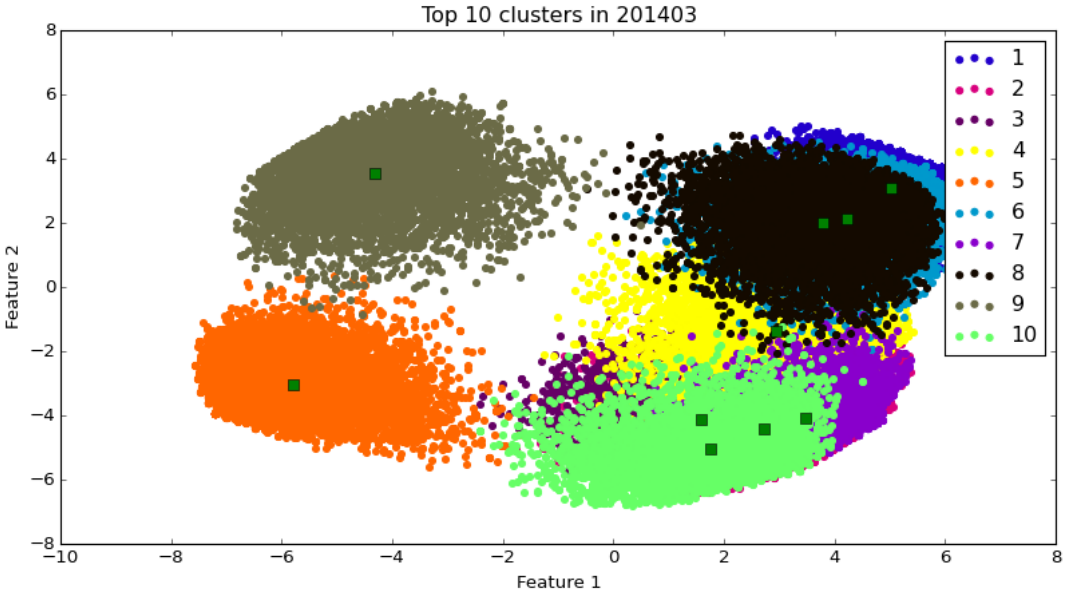

Figure 3: Top 10 clusters in March 2014

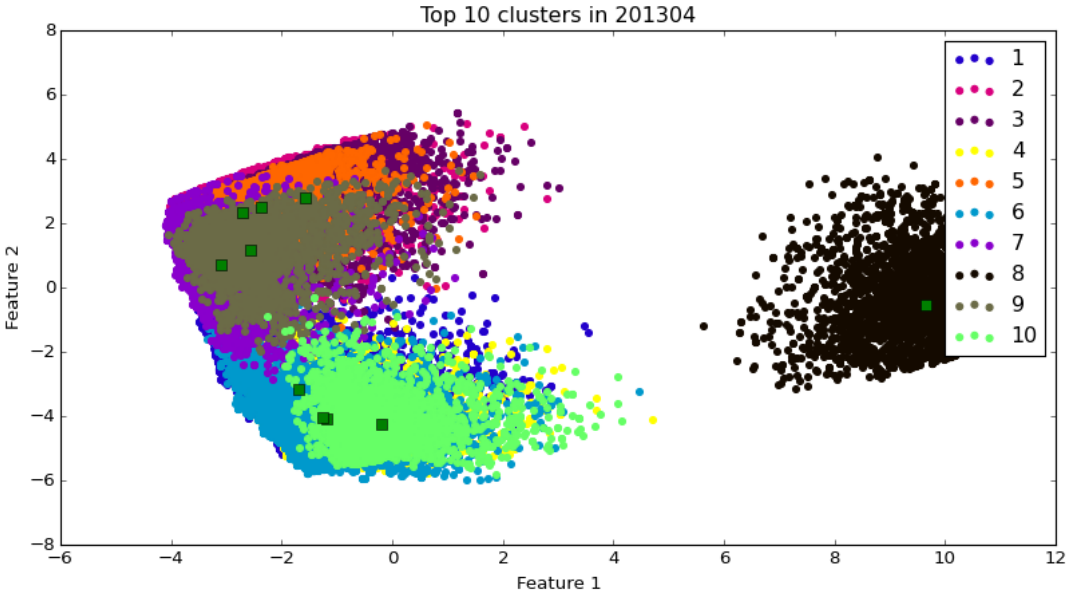

Figure 4: Top 10 clusters in April 2013

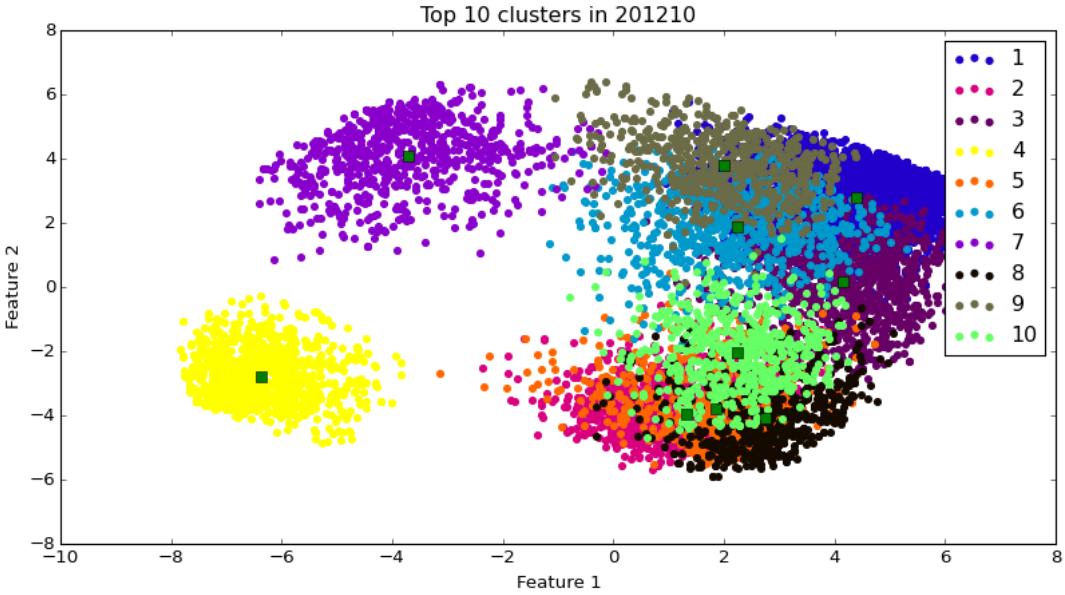

Figure 5: Top 10 clusters in October 2012

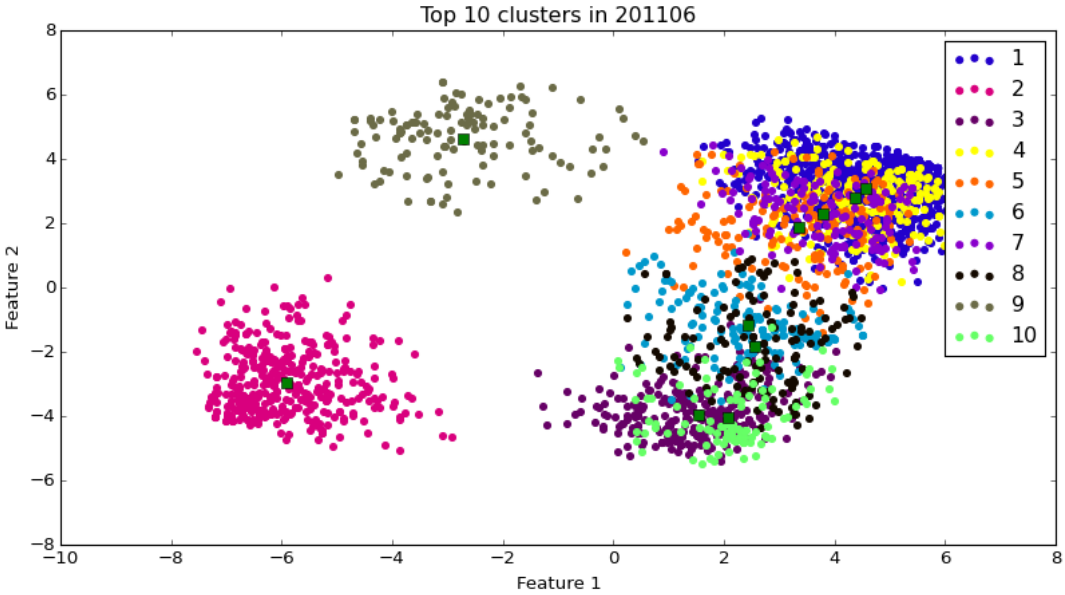

Figure 6: Top 10 clusters in June 2011

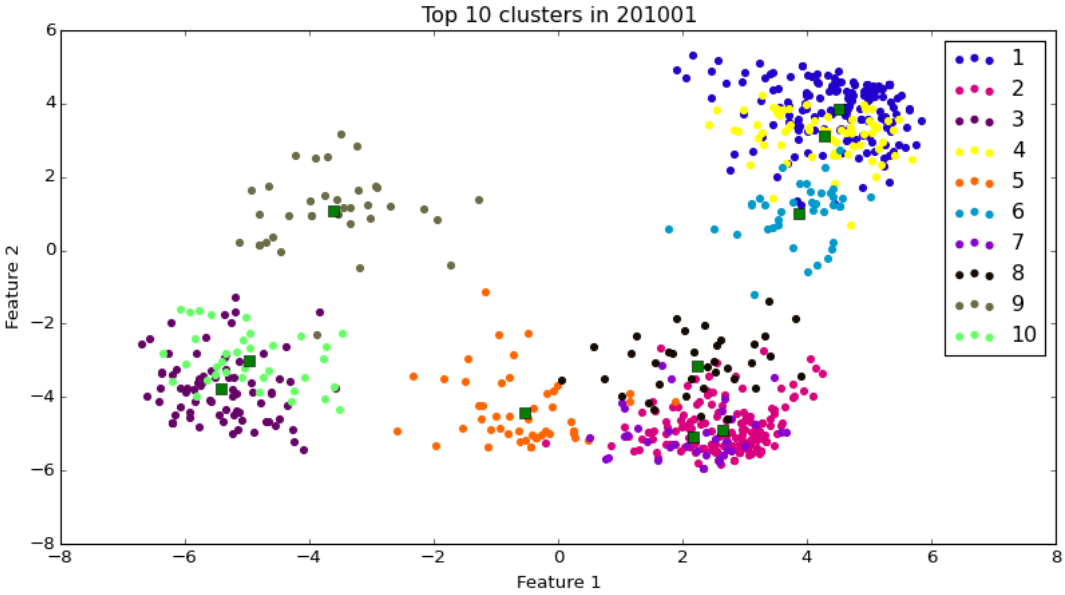

Figure 7: Top 10 clusters in January 2010

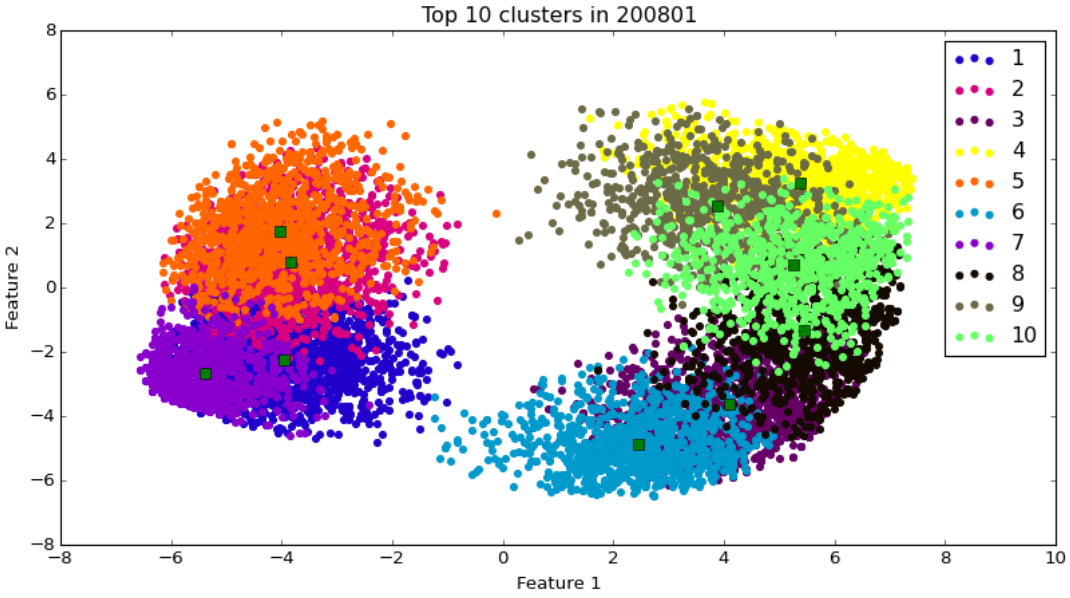

Figure 8: Top 10 clusters in June 2008

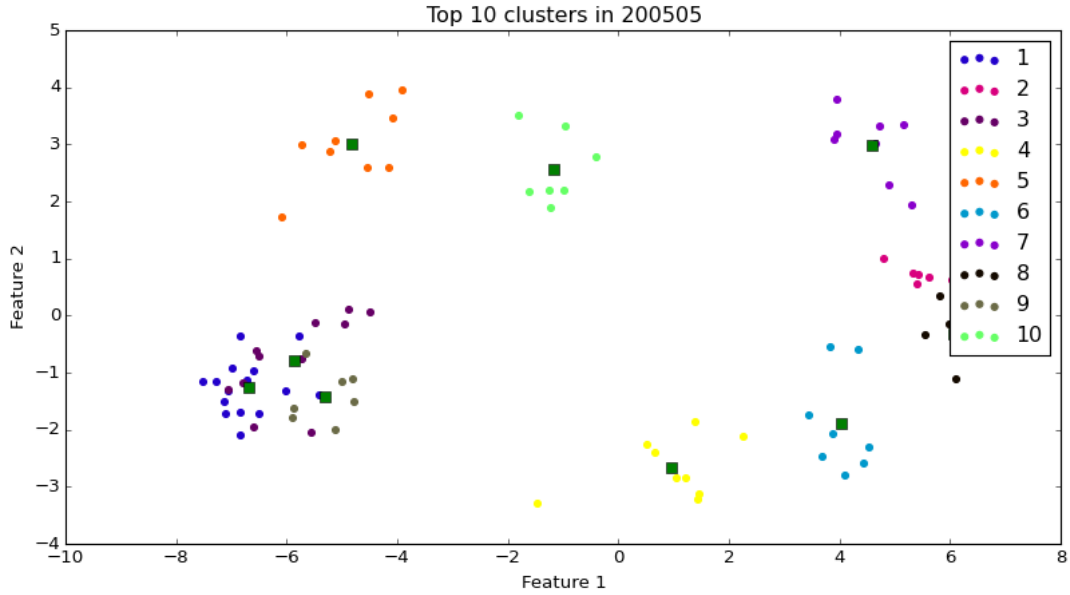

Figure 9: Top 10 clusters in May 2005

From the above images, it is clear that there are some inherent clusters within the images in a specific time period.

And also note that from the above images, we can see lots of overlapping, which is caused by the projection of 64 dimension data to 2 dimension, and does not mean the real clusters are heavily overlapped.

Then we also draw some figures to show the distribution of the sizes of the clusters. Here are some examples:

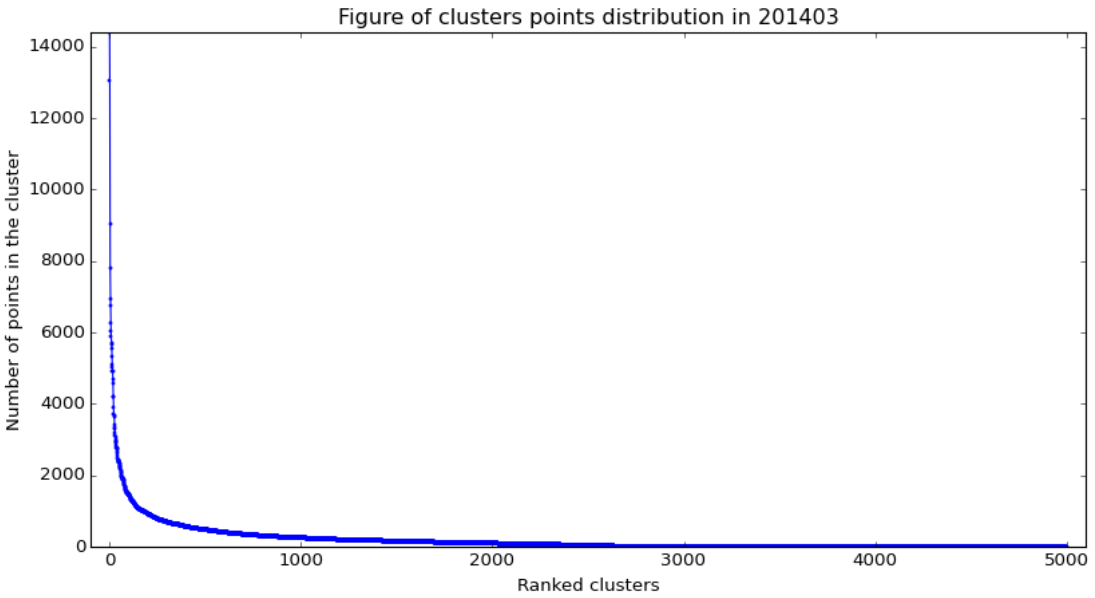

Figure 10: Clusters size distribution in March 2014

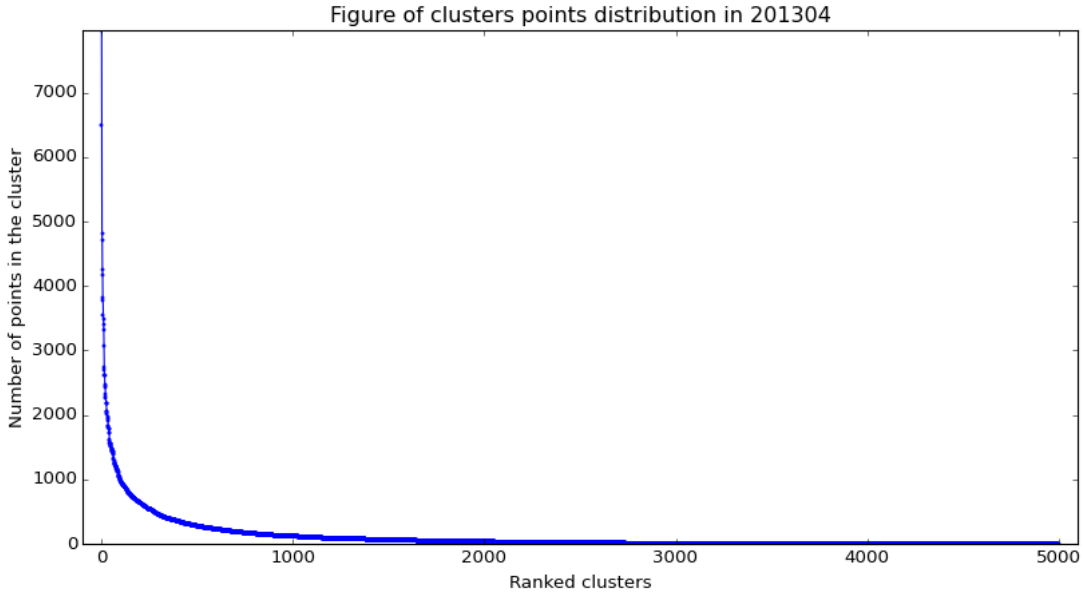

Figure 11: Clusters size distribution in April 2013

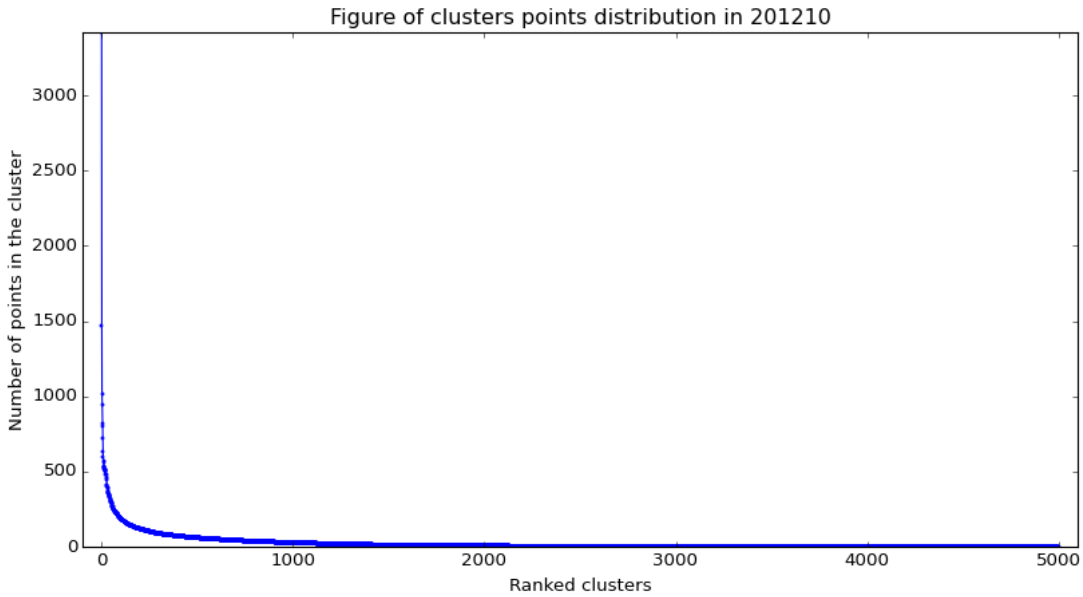

Figure 12: Clusters size distribution in October 2012

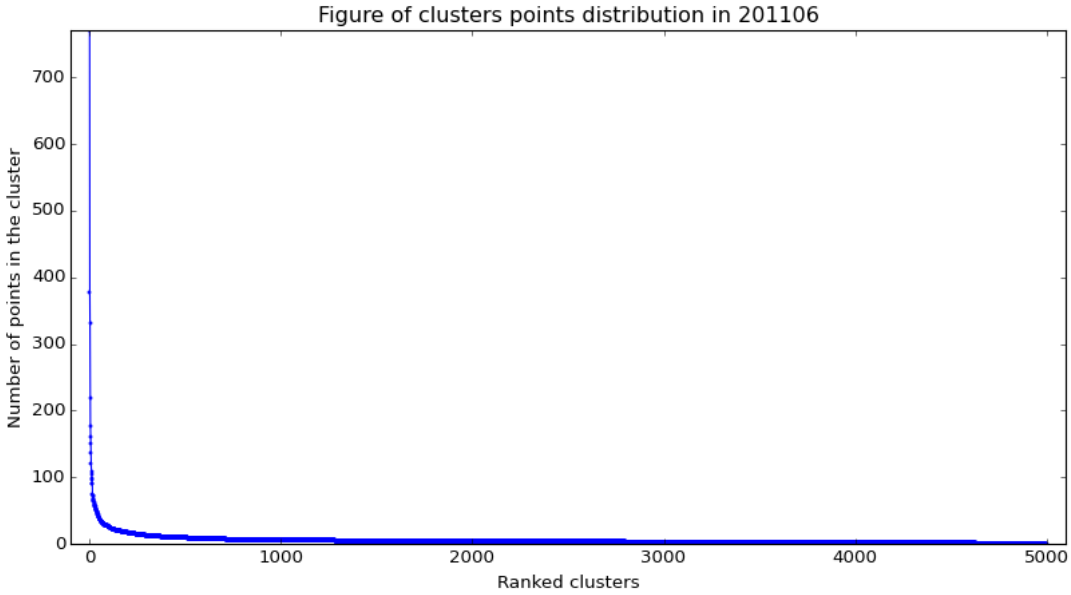

Figure 13: Clusters size distribution in June 2011

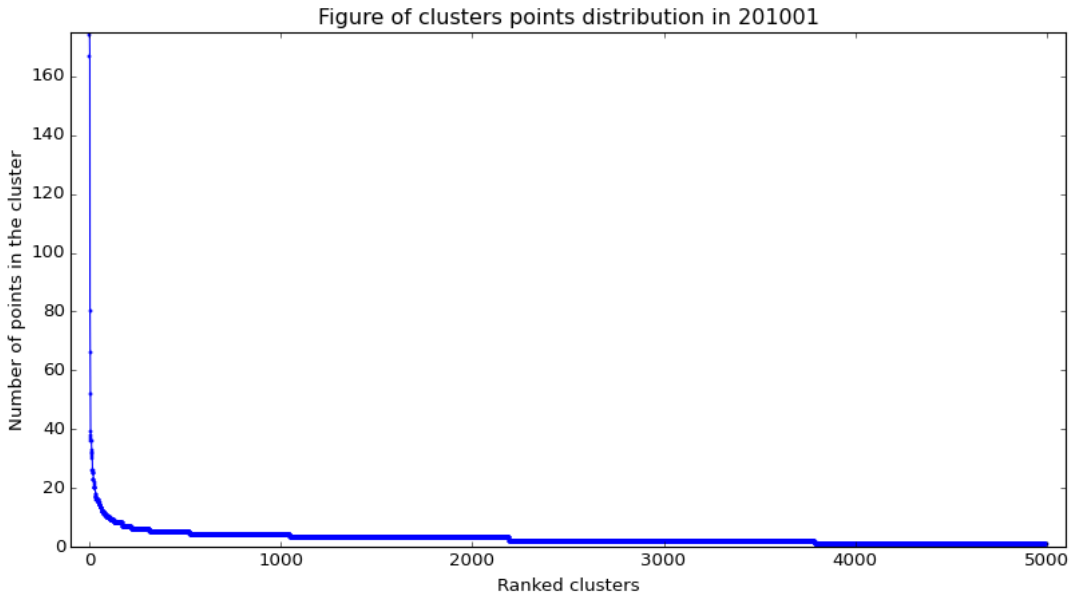

Figure 14: Clusters size distribution in January 2010

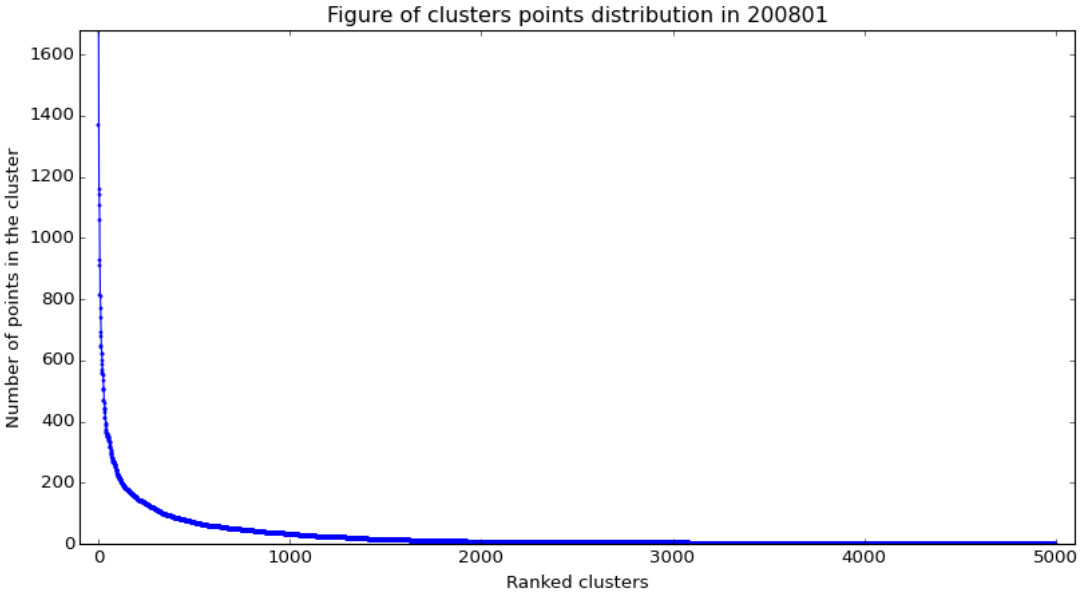

Figure 15: Clusters size distribution in June 2008

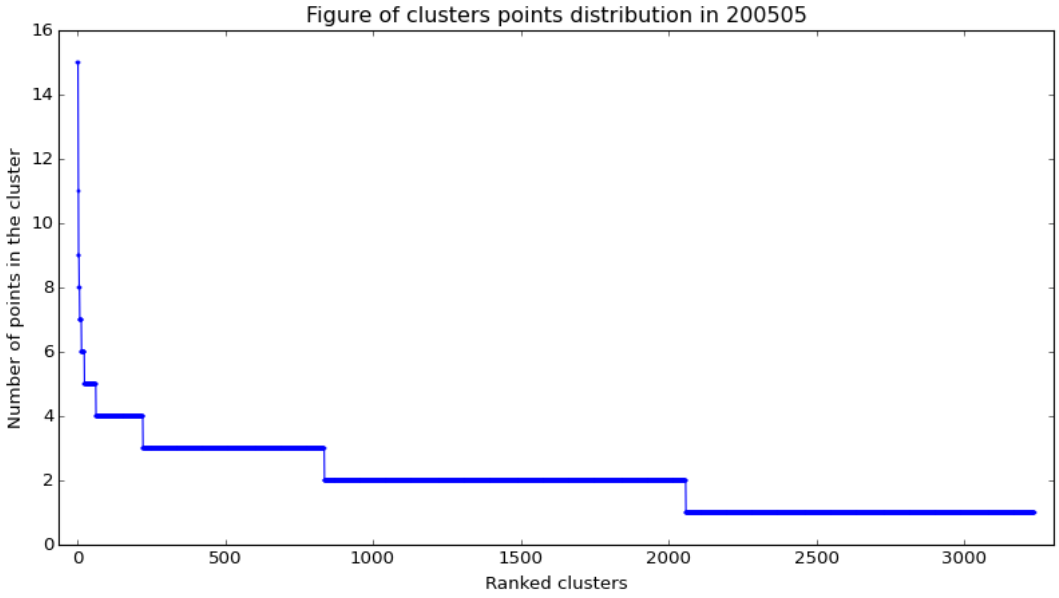

Figure 16: Clusters size distribution in May 2005

From the above images, we can see that the size of the clusters does not have a long tail.

4 Conclusion & Discussion

If we look at possible improvements for these cases, we find the following problems that should be solved.

4.1 How to get better image features?

In the workflow we have demonstrated, the result of aggregation largely depends on the features (hash code) that are extracted by the average hash algorithm or pHash algorithm. These algorithms are designed to find images that are similar. But we have to ask several questions: are the images considered similar by either of these 2 algorithms really similar? Does this kind of similarity actually make sense to human beings?

Actually both the average hash algorithm and the pHash algorithm are the most naive and intuitive methods to extract features from the images. And there are lots of other good image feature extraction methods, like GIST, Spectral Hashing, and so on. But all these methods do not have a Java implementation, and it is not so easy to implement them by ourselves. But trying other image descriptors can be a potential future step to improve the results.

4.2 How to do clustering efficiently in big data?

Essentially, this is the key problem we are working on. The solution we use is firstly reduce the data dimensions (image hash code extraction), and then use a time and memory saving clustering algorithm (K-means).

The Apache provide a scalable machine learning library, Mahout, which contains several useful clustering algorithm. But now the Mahout is still in version 0.9, and not so stable. We have tried the clustering algorithm in Mahout, and the results show that it is extremely slow in clusters initialization step. So it is useful if the user just needs a few clusters, and in our case, we need lots of clusters (for example, 5000 or more), and the Mahout is very slow (from our testing, even slower than running similar algorithms locally).

4.3 This is not a real-time workflow.

We have built a demo that demonstrates the idea, which is similar to "Google Trends" but in the form of images. Unfortunately, while Google Trends is real time, the workflow that we have shown cannot be easily migrated to a real-time context. The main issue here is that the workflow itself is not real-time. In principle when new data comes in, the whole process has to be started from the beginning (at least from the beginning of the month). One possible solution is compare the hash code of images of incoming data with those of existing data entries. Rerun the process only when it seems possible for the new data to influence the current result.

Despite of this problem, the workflow we have demonstrated is still useful especially in the context of static data, for instance, the data set is stable, or the user only need to know the trends once a day or once a week.