PubMed Visaulization

Anirudh Negi, Junfeng Zhao, Yibo Zhou

November 8, 2021

Abstract

Currently there are millions of published articles and papers stored in a massive database for all articles published in medical journals since the 1960s. However, not only is it difficult to search through these articles, the sheer volume of data makes gathering insights next to impossible. We propose a hopefully permanent solution to this issue by creating visualizations of all the data that can be updated in a streaming context. The primary techniques to extract these visualizations will be clustering and ranking.

1 Goals

1.1 Questions

- Can we rank the "trustworthiness" of attributes like author, journal or article using PageRank?
- Can we visualize associations among attributes like author, journal or articles?
- Can we process a massive dataset (30 million rows, $>$ 10GB) efficiently using a streaming context? What will be the major challenges here?

1.2 Data Description

"PubMed[5] is a free resource supporting the search and retrieval of biomedical and life sciences literature with the aim of improving health–both globally and personally.

The PubMed database[6] contains more than 33 million citations and abstracts of biomedical literature. It does not include full text journal articles; however, links to the full text are often present when available from other sources, such as the publisher's website or PubMed Central (PMC).

Available to the public online since 1996, PubMed was developed and is maintained by the National Center for Biotechnology Information (NCBI), at the U.S. National Library of Medicine (NLM), located at the National Institutes of Health (NIH)." [5]

- Total number of files: 1062
- Original data size(.xmls) 150-200GB
- Location: https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/ [6]

After Conversion to csv:

- Total citations: 31,851,988 (31.8 Million approx) (9GB approx)
- Total edges between them: 214,169,935 (214.2 Million approx) (2GB approx)

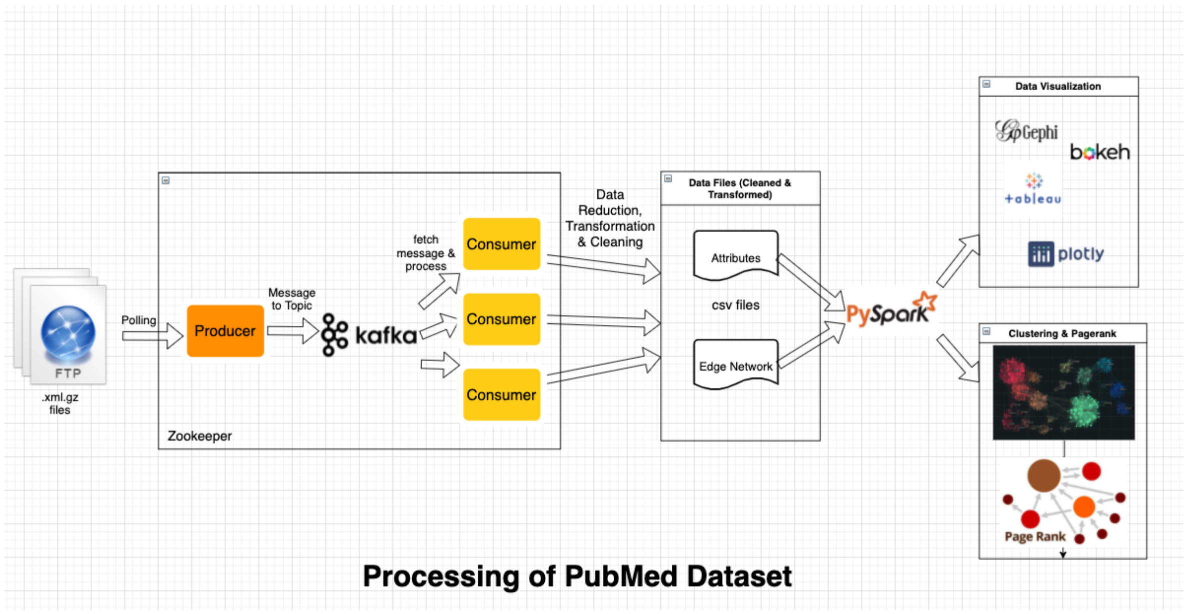

Figure 1: This is our planned data workflow

2 Tools, Technologies, and Libraries

For Data Processing, we set up Apache Kafka[1] single node cluster with zookeeper and created a topic that holds the urls for each file present on the ftp. The producer and consumer are written with python, so we used the kafka library for sending and listening messages respectively. For parsing xml data to csv, we used the xml.etree.ElementTree module and pandas and pyspark[7] libraries for data cleaning, transformation and generating graph data.

For analysis, we used nltk, numpy, bokeh[2], matplotlib, gensim, spacy, pyclustering[8] and sklearn libraries containing different modules like TfidfVectorizer for feature extraction from the article titles, PCA (Principal Component Analysis) for feature reduction, cure for implementing clustering on the features extracted, TSNE for dimension reduction of multidimensional vector to 2 dimensions and cluster_visualizer for cluster representation on 2D plane.

2.1 Data Processing

PubMed data exists in ftp location in xml.gz format, which requires transformation into csv, enabling it to be used in the environments and language we are using i.e. python, pyspark, pandas, etc. Moreover, there are various data elements that need not be required for the solution we are building on the desired questions to be solved. Another challenge in data processing is that the data is distributed across multiple files instead of a single file and it occupies more than 200GB storage. So, to solve this issue, we are implementing kafka streaming where the producer polls on the ftp location for any unprocessed files and sends the message to a kafka topic and the consumers listen to these topics and process individual files. The processing involves reading the xml.gz file, parsing it to csv, data cleaning, reduction and transformation. Apart from this, consumers also create csv files for the edge network that is found in the PubMed citation dataset. This is useful to implement graph algorithms.

3 Data Analysis and Visualization

3.1 Data Files

pubmed21n0173.xml.gz: Original file that contains the xml data elements.

- 21 represents the last two digits of the baseline year.
- 0173 is the file number. For the year 2021 there are 1062 total files.

pubmed21n0173.csv: csv file produced by consumer, which contains attributes of PubMed citations required for our analysis.

pubmed21n0173_graph.csv: csv file produced by consumer, which contains citation id mapped to all the reference citations.

3.2 Planned Visuals

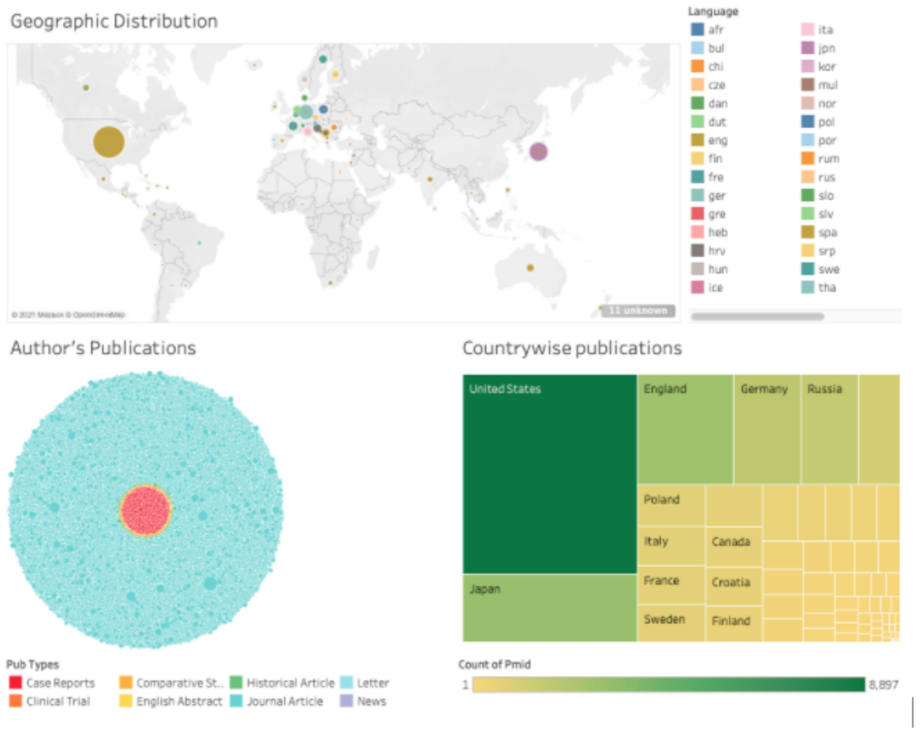

Figure 2: PubMed Data Visualization Dashboard

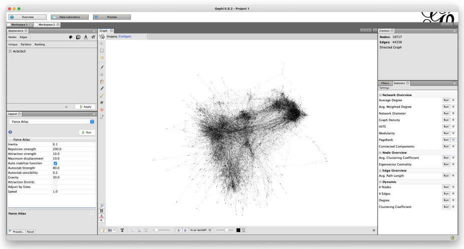

Figure 3: Sample network for PageRank in Gephi[3]

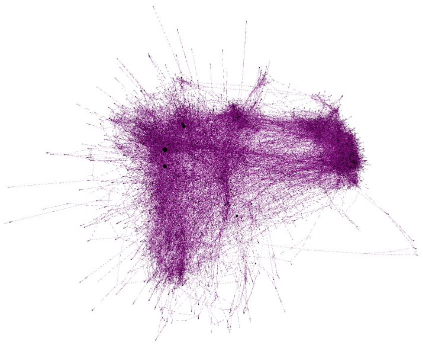

Figure 4: Sample network of citations in Gephi

4 Clustering

4.1 TSNE

T-distributed stochastic neighbor embedding is a popular method for visualizing document similarity. Its main objective is dimensionality reduction, decomposing high-dimensional vectors to 2D space. We implemented this on our sample dataset using article titles. The process starts from splitting the text into words, building bigram and trigram models, removing stop words, creating a dictionary, followed by calculating term frequencies and building an LDA (latent Dirichlet allocation) topic model. In LDA models, each article is composed of multiple keywords, but typically one is dominant. So, with weights assigned, we can identify the dominant keywords and using the TSNE model, we can fit and represent this data into a 2D space. This technique however, is very expensive so typically SVD or PCA is applied ahead of time.

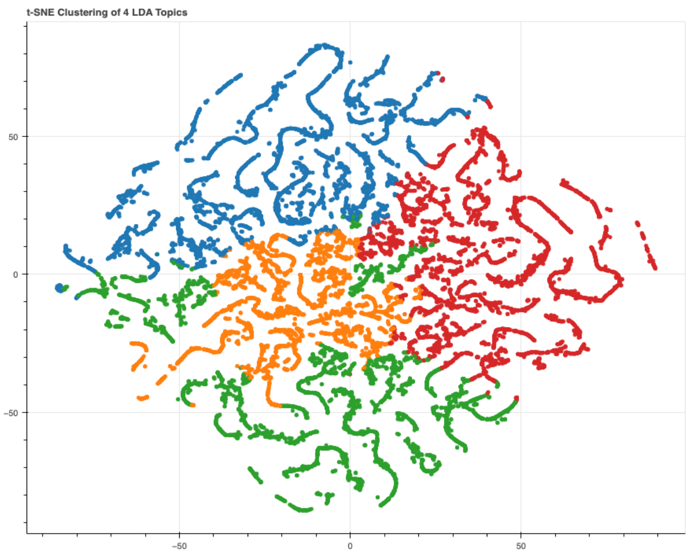

Figure 5: t-distributed stochastic neighbor embedding

4.2 Text clustering based on MiniBatchKMeans

MiniBatchKMeans is a variant of KMeans algorithm which uses mini-batches to reduce computation time, while trying to optimise the objective. It uses a subset of data as batches (randomly sampled) in each training iteration drastically reducing the amount of computation required to converge to a local solution. While our objective is to use a clustering module, we used MiniBatchKMeans module from sklearn.cluster. We split the dataset based on publication type into multiple files, then we assign label(publication type number starting from 0) to each article title, then we vectorize the article titles using TfidfVectorizer from sklearn.feature_extraction.text. Using the MinibatchKMeans module, we get the cluster centers in vectorized form as well. So, basically, we get vector data for each article, which we decompose (including the cluster centers) into 2 dimensions using PCA module sklearn.decomposition. Once we have the reduced features, we represent it as a 2D scatter plot.

Algorithm 1 Clustering with MinibatchKMeans

```
MinibatchKMeans (k, mini−batch size, iterations)
The algorithm iterates between two steps, similar to k−means.
1. In the first step, samples are drawn randomly from the dataset, to form
   a mini−batch. Then these are assigned tothe nearest centroid.
2. In the next second step, the centroids are updated. In contrast to
   k−means, this is done on a each sample basis.
   For each sample in the mini−batch, the assigned centroid is updated by
   taking the streaming average of the sample and all previous samples
   assigned to that centroid.
```

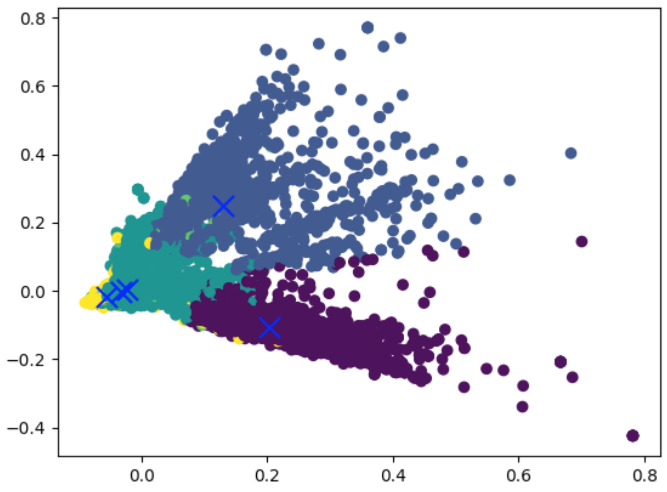

Figure 6: Text Clustering with MiniBatchKMeans

4.3 Improvising text clustering using CURE

One thing we observed with MiniBatchKMeans clustering on our dataset was that it gives us overlapping clusters, and the clusters centers are also very near, it also gives a homogeneity score of 0.03 which is very close to 0 which also indicates overlapping clusters. The other issue is, we tested on a small dataset, but what if we have a humongous dataset? Well, in that case, we have to switch to an algorithm that is well suited for large datasets. So, we planned to use the CURE algorithm for clustering, improvising the previous approach. CURE(Clustering Using REpresentatives) is an efficient algorithm for clustering large datasets. As compared to K-means or its variations, it is more robust to outliers. The Pyclustering library has a cure module which helps us to get clusters, get the cluster representatives and the means and with the cluster_vizualizer, we can represent our clusters. It solves the issues that we get using MiniBatchKMeans.

Algorithm 2 Clustering with CURE

```
CURE (no. of points,k)
Input : A set of points S
Output : k clusters
1. For every cluster u (each input point), in u.mean and u.rep store
   the mean of the points in the cluster and a set of c representative
   points of the cluster (initially c = 1 since each cluster has one
   data point). Also u.closest stores the cluster closest to u.
2. All the input points are inserted into a k–d tree T
3. Treat each input point as separate cluster, compute u.closest for
   each u and then insert each cluster into the heap Q. (clusters are
   arranged in increasing order of distances between u and u.closest).
4. While size (Q) > k{
    a. Remove the top element of Q (say u) and merge it with its closest
    cluster u.closest (say v) and compute the new representative points
    for the merged cluster w.
    b. Remove u and v from T and Q.
    c. For all the clusters x in Q, update x.closest and relocate x
        i. insert w into Q
    }
5. repeat
```

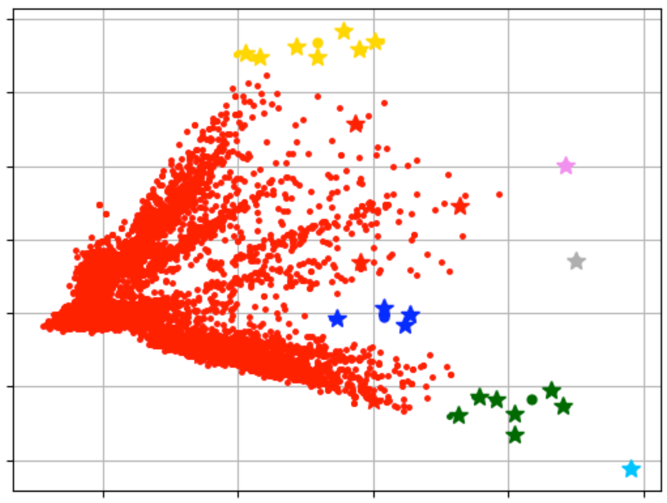

Figure 7: Text Clustering with CURE Algorithm

5 PageRank

Using pagerank[4], we attempt to provide a solution for measuring the importance of scientific papers in the field of medicine. We can say it is an extension to an integer counting algorithm that would count the number of citations. It can be used to enhance the counting using a more integrated picture of citations' influence based on the references within each citation. This is a very useful algorithm that would help us to answer most of our questions. Not only the importance, but it can also help us rank the trustworthiness of an article, an author or even a journal. Given its recognized benefits in quantifying relative importance, it can enrich the citation network, thereby overcoming the existing inadequacy of citation counts alone.

Algorithm 3 Pagerank with Spark Streaming

```
def computeContribs(neighbors, rank):
    for neighbor in neighbors:
        yield (neighbor, rank / len(neighbors))

def main(input_folder_location):
    sc = SparkContext.getOrCreate()
    # Streaming will execute in each 20 seconds
    ssc = StreamingContext(sc, 20)
    lines = ssc.textFileStream(input_folder_location)

    counts = lines.map(lambda line: line.split(",")).map(lambda pages:
    (pages[0], pages[1])).transform(lambda rdd: rdd.distinct()).groupByKey().map(lambda x:
    (x[0], list(x[1])))

    ranks = counts.map(lambda element: (element[0], 1.0))
    contribs = counts.join(ranks).flatMap(lambda row: computeContribs(row[1][0], row[1][1]))
    ranks = contribs.reduceByKey(lambda v1, v2: v1 + v2)
    ranks.pprint()
    ranks.saveAsTextFiles('out')
    ssc.start()
    ssc.awaitTermination()
```

6 Observations

With Mini Batch KMeans, we observed that the algorithm starts very well, with small data, however, as the more data comes in, there is decrease in homogeneity, i.e. there are overlapping clusters. Also, the algorithm takes more time to process large data set.

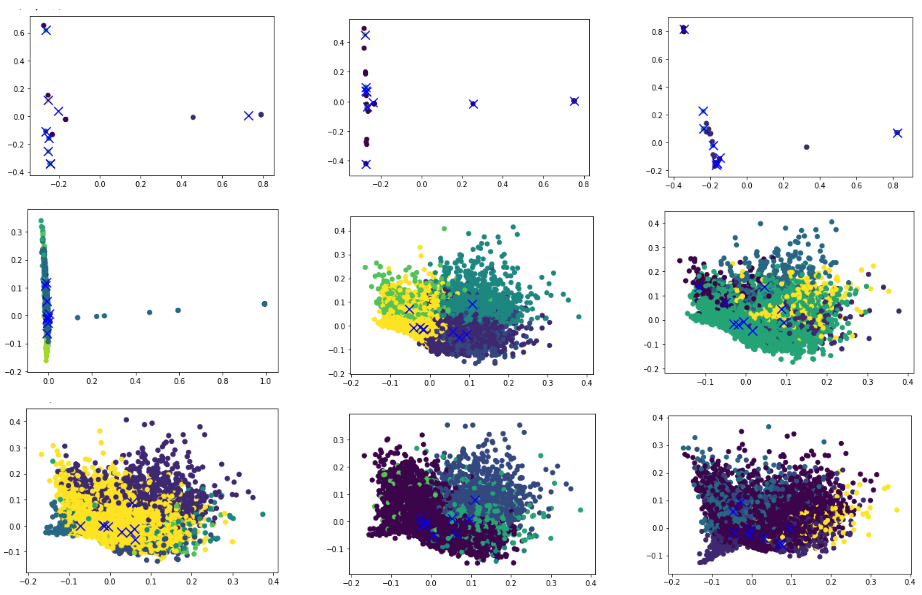

Figure 8: Processing of miniBatchKMeans

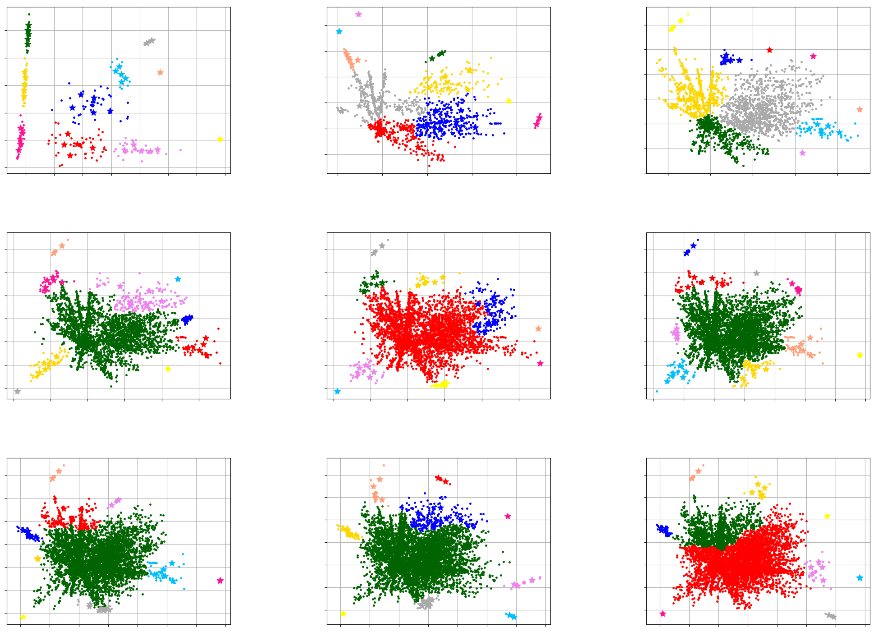

Figure 9: Processing of CURE

Using CURE algorithm, the issue of handing large dataset is resolved since we are considering sample data points with representers in each iteration. We also observe that there is no overlapping of clusters with CURE Algorithm.

7 Conclusion

Coming back to the original questions we were able to rank the articles using pagerank algorithm based on the edge network that we were able to generate using id(s) of pubmed articles with id(s) of other pubmed articles referenced in each article. With this ranking and actual number of articles published in a particular journal and number of articles by each author, we were able to identify if the articles published by a particular author in a particular journal could be helpful in a related research. We were also able to classify articles under specific topics using tSNE which is a useful approach for document similarity. We also compared mini-Batch KMeans and CURE Algorithms on large dataset and concluded that CURE is a much better approach on large datasets. We are able to process data using kafka streaming for data processing, and spark streaming for page rank, and using CURE Algorithm, we could process large dataset for clustering.

8 Future Work

Citation analysis is a well-researched field and we would like to not only help further its progress but all see where our techniques stack up. Currently, we are still exploring but have found a variety of tools available from UIC. [9]

9 Timeline

This figure shows the timelines for our project including division of work represented with different colors.

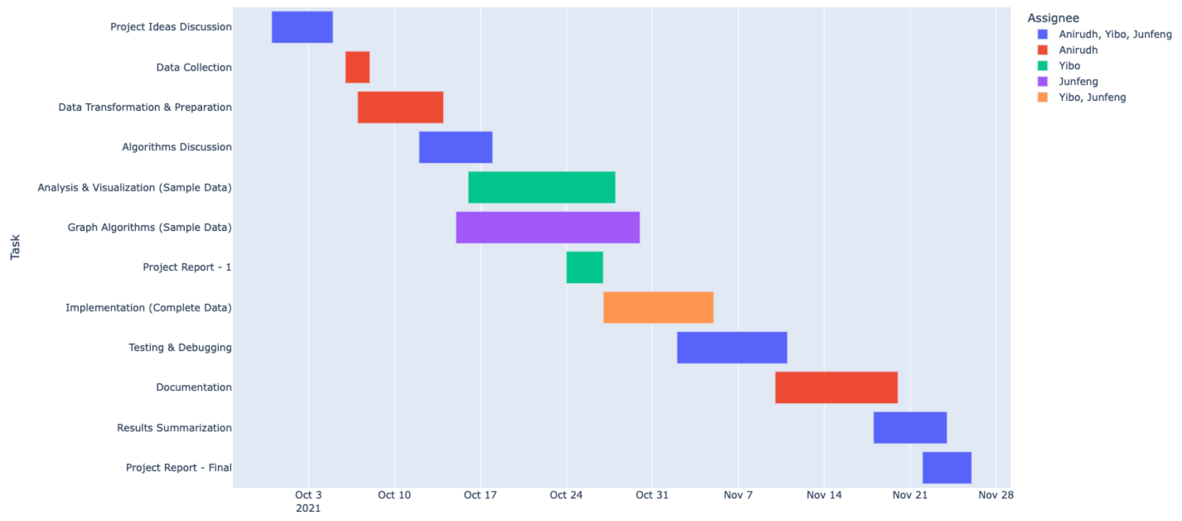

Figure 10: This is our planned timeline

References

REFERENCESREFERENCES

[1] Kiruparan Balachandran. *QuickStart — Apache Kafka + Kafka-Python*. 2019. URL: https://towardsdatascience.com/quickstart-apache-kafka-kafka-python-e8356bec94.

[2] Bokeh. *Bokeh*. 2021. URL: https://docs.bokeh.org/en/latest/.

[3] Gephi. *Gephi*. 2021. URL: https://gephi.org/.

[4] Manning. *Mastering Large Datasets*. 2021. URL: https://livebook.manning.com/book/mastering-large-datasets/chapter-9/45.

[5] NLM. *PubMed About*. 2021. URL: https://pubmed.ncbi.nlm.nih.gov/about/.

[6] NLM. *PubMed Baseline*. 2021. URL: https://www.nlm.nih.gov/databases/download/pubmed_medline_documentation.html.

[7] Oracle. *GraphX Programming Guide*. 2021. URL: http://spark.apache.org/docs/latest/graphx-programming-guide.html.

[8] PyClustering. *PyClustering*. 2021. URL: https://pyclustering.github.io/docs/0.8.2/html/index.html.

[9] UIC. *Measuring Your Impact: Impact Factor, Citation Analysis, and other Metrics: Citation Analysis*. 2021. URL: https://researchguides.uic.edu/c.php?g=252299&p=1683205.