Remembering the Forgotten: Clustering, Outlier Detection, and Accuracy Tuning in a Postdiction Pipeline

Anna Baskin[1], Scott Heyman[1], Brian T. Nixon[1](✉), Constantinos Costa[2], and Panos K. Chrysanthis[1]

[1] Department of Computer Science, University of Pittsburgh, Pittsburgh, USA
{afb39,sth66}@pitt.edu, {nixon.b,panos}@cs.pitt.edu

[2] Rinnoco Ltd, 3047 Limassol, Cyprus
costa.c@rinnoco.com

Abstract. The ever-increasing demand to use and store data in perpetuity is limited by storage cost, which is decreasing slowly compared to computational power's exponential growth. Under these circumstances, the deliberate loss of detail in data as it ages (referred to as data decay) is useful because it allows the cost of storing data to decrease alongside the data's utility. The idea of data postdiction as a data decay method uses machine learning techniques to recover previously deleted values from data storage. This paper proposes and evaluates a new pipeline using clustering, outlier detection, machine learning, and accuracy tuning to implement an effective data postdiction for archiving data. Overall, the goal is to train a machine learning model to estimate database features, allowing for the deletion of entire columns, which can later be reconstructed within some threshold of accuracy using the stored models. We evaluate the effectiveness of our postdiction pipeline in terms of storage reduction and data recovery accuracy using a real healthcare dataset. Our preliminary results show that the order in which outlier detection, clustering, and machine learning methods are applied leads to different trade-offs in terms of storage and recovery accuracy.

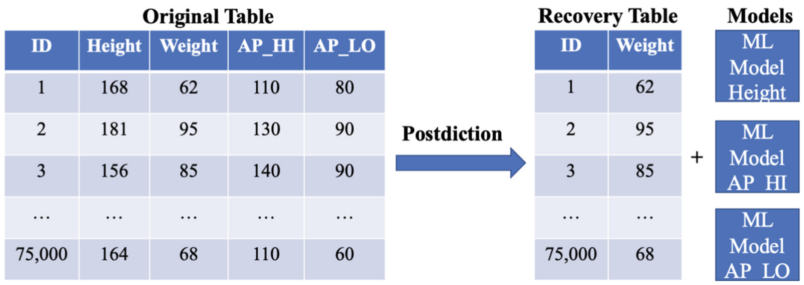
Keywords: Data postdiction · Data Decaying · Lossy Compression · Clustering · Outlier Detection

1 Introduction

Enterprises are generating data at unprecedented rates as more services are hosted online and IoT devices are widely deployed. From government, to industry, to entertainment and healthcare, all enterprises are collecting and exploiting this data for business intelligence, and online decision making. For example, the National Institutes of Health (NIH) initiated the *All of Us* research program [6] with the goal of establishing one of the most diverse health databases

A. Baskin, S. Heyman, B. T. Nixon—These authors contributed equally

A. Abelló et al. (Eds.): ADBIS 2023, CCIS 1850, pp. 46–55, 2023.
https://doi.org/10.1007/978-3-031-42941-5_5



ID	Height	Weight	AP_HI	AP_LO
1	168	62	110	80
2	181	95	130	90
3	156	85	140	90
...	...	...	...	...
75,000	164	68	110	60

ID	Weight
1	62
2	95
3	85
...	...
75,000	68

Fig. 1. Example of Basic Postdiction on Medical Data

ever created for precision medicine and disease prediction. Also, telecommunication companies are collecting data that can be used from churn prediction of subscribers, city localization, 5G network optimization and user-experience assessment [3]. The demand to support many analytic-oriented processing scenarios using larger collections of data that span over a longer period of time has led to an ever-increasing demand in storing big data in perpetuity.

The continuous storage of big data has the potential to facilitate extensive analysis. Furthermore, the rapid growth of computational power allows such analysis to be performed at faster rates. However, the expenses associated with storing such a vast amount of data is increasingly becoming a constraining factor. The storage cost is decreasing at a much slower rate compared to the exponential growth of computational power. This is not a new challenge and significant research has been done into *lossless compression* as a solution to this problem as well as into *data reduction* techniques [12], e.g., sampling [11], aggregation (OLAP) [8], dimensionality reduction (LDA, PCA) [7], synopsis (sketches) [1] and lossy compression [9]. Only recently the focus was shifted on utilizing *data decay* [4,5], i.e., the deliberate loss of detail in data as it ages, as it allows the cost of storing data to decrease alongside the data's usefulness.

Data postdiction [2,3] is a new way to implement data decay making a statement about the past value of some tuple, which does not exist anymore as it had to be deleted to free up disk space. Data postdiction relies on existing Machine Learning (ML) algorithms to abstract data into compact models that can be stored and queried when necessary [1–3]. This allows for the deletion of entire data columns, as shown in Fig. 1, where one or more columns are saved to be able to recover the postdicted columns using the stored models.

Postdiction currently considers the complete dataset to be decayed similar to compression. In contrast to compression, postdiction can retrieve individual data values without recovering the entire compressed dataset. Similar to lossy compression, postdiction exhibits a *storage-accuracy* trade off. The goal of this work is to improve storage reduction and data recovery accuracy by exploiting data partitioning.

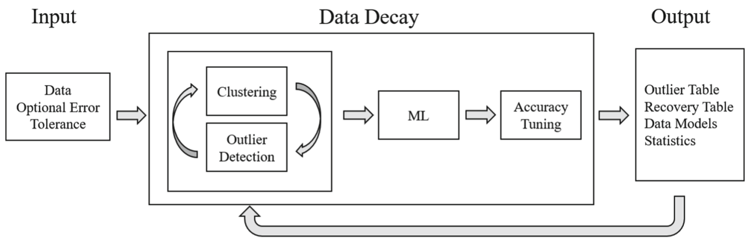


Fig. 2. Data Postdiction Pipeline

This paper hypothesizes that *clustering*, *outlier detection*, and *accuracy tuning* can be used to implement data partitioning that improves the original postdiction effectiveness. The contribution of this paper is a novel postdiction pipeline that explores these methods to produce the best possible data storage reduction for a specified *error tolerance*, i.e., data recovery accuracy (Sect. 2).

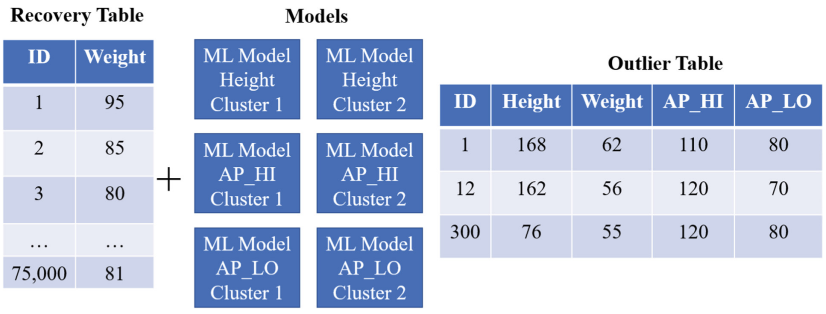
We use our postdiction pipeline to observe the effects of various combinations of clustering, outlier detection using LSTM (Long Short-Term Memory) to generate the machine learning models, with the goal of observing how the three strategies of clustering, outlier detection, and accuracy tuning can be used together to create better postdiction (Sect. 3).

2 Our Approach: Postdiction Pipeline

In our approach, we extend the original postdiction methodology, which uses only machine learning for data decay (Fig. 1), into a postdiction pipeline using clustering, outlier detection, and accuracy tuning, as shown in Fig. 2, to achieve improved postdiction results. The improved results take two forms: (1) the best reduction balance between data storage and recovery error (similar to the original postdiction), and (2) the best reduction in data storage given a specified data recovery accuracy/error tolerance.

Workflow. In the pipeline, input data is passed first into clustering or outlier detection. They can be run in any order: outlier detection before clustering, after clustering, or before and after clustering. Each of the generated clusters are then used to train a machine-learning model. At this stage the pipeline produces a recovery table, the machine learning models and a table of outliers, representing the best achievable storage reduction and recovery accuracy, without any specified error tolerance threshold.

Next, if the error tolerance is provided, accuracy tuning is used to predict each value in the recovery table, pruning any results outside a designated error threshold and placing them in the outlier table alongside the previously detected outliers. At the end, the pipeline outputs the final table of outliers, the recovery table, the machine learning models, and statistics relating to the number of outliers and the recovered accuracy.



Recovery Table

ID	Weight
1	95
2	85
3	80
…	…
75,000	81

+

Models

ML Model Height Cluster 1	ML Model Height Cluster 2
ML Model AP_HI Cluster 1	ML Model AP_HI Cluster 2
ML Model AP_LO Cluster 1	ML Model AP_LO Cluster 2

Outlier Table

ID	Height	Weight	AP_HI	AP_LO
1	168	62	110	80
12	162	56	120	70
300	76	55	120	80

Fig. 3. Output of Our Postdiction Pipeline

Ideally, we would like to decay the maximum number of columns using the minimum number of features. An ideal sample output of the pipeline including the recovery table, machine learning models, and outlier tables is shown in Fig. 3. The example assumes that `Weight` best predicts all other fields/columns of the original table (shown in Fig. 1) and that each field can be partitioned into two clusters, possibly using different clustering methods, each of which could be compacted into a single model using different ML algorithms.

Rationale. The underlying reasons for each of the four component of our postdiction pipeline are:

- *Outlier detection:* Given that a dataset's distribution is unknown a priori, outlier detection is important for postdiction for a couple of reasons. Machine learning models typically struggle to predict outliers, but their very presence in the dataset deteriorates model performance. Therefore, if the outliers are removed and stored before running the machine-learning model in postdiction, users will be able to recover outliers with one hundred percent accuracy and have a better model for recovering the rest of the data. However, storing the outliers is a trade-off from traditional postdiction. On one hand, more storage space is required to maintain the outliers, but storing them means that more data can be recovered accurately.
- *Clustering:* Clustering algorithms group data in such a way that the data in each group are more similar to each other than to data in other groups. Clustering is important for postdiction because it can make machine learning models perform better. Having data that are all similar and close together makes it easier to recover later at a higher accuracy. Furthermore, if the algorithm finds multiple clear and distinct clusters of data, the model will have clear decision thresholds, which will allow for more accurate recovery. Recovering the data, however, will now require storing a model for each cluster, which can increase storage costs.
- *Machine Learning:* Any machine learning algorithm can be used in our pipeline to generate the compact ML models, for example, any type of recurrent neural network or logistic regression. The choice could be determined by the type of the data.

– *Accuracy Tuning:* As the final step in the pipeline, accuracy tuning is used to guarantee an error boundary. Accuracy tuning allows the user to set an error threshold, specifying the exact amount of error they are willing to tolerate. Continuous variable predictions beyond the error threshold are added to the outlier table, alongside incorrect categorical variable classifications. This is a very powerful tool because it gives the user more control over how the data decays. Without this accuracy tuning, users have no guarantee as to the accuracy of any individual recovered data point.

3 Experimental Evaluation

In our experimental evaluation, we compare the results from combinations of outlier detection algorithms, clustering algorithms and postdiction workflows.

3.1 Experimental Setup

Algorithms. We selected a representative sample of outlier detection and clustering algorithms, which worked in fundamentally different ways, as described below, to observe their effectiveness in the postdiction pipeline. These were implemented using the *scikit-learn* library in Python.

– *Outlier Detection:* The methods of outlier detection chosen for experimentation were Z-score and Isolation Forest. Z-score is a basic statistics measurement describing how many standard deviations a single point in the data is above or below the mean of the data. Z-score outlier detection was chosen for its simple and consistent results. Any data point greater than three times the standard deviations from the mean is considered an outlier. Isolation Forest outlier detection was chosen because of its ability to be applied to very large data sets. Rather than profiling all data points to calculate the outliers (as in Z-score), Isolation Forest is based on the principle of detecting only anomalies. This makes it faster than Z-score on large datasets.
– *Clustering:* The clustering methods chosen for experimentation were K-means, Density-based spatial clustering (DBSCAN), and Gaussian Mixture Modelling (GMM). These are some of the de facto methods of clustering, which is one of the main reasons for choosing them. Additionally, all of these methods work in different ways and can lead to different results. This is deliberate in order to maximize the chance of getting results that are not dependent on which clustering algorithm is chosen. Something important to note is that DBSCAN does not take in as input the number of clusters desired for output.
– *Machine Learning:* We experimented with LSTMs as they are capable of processing both single data points and sequences of data. While the dataset used by this paper does not contain sequential data, postdiction was originally concocted as a method of reducing the size of time-series telecommunication data [3]. The LSTM models were built using the *Keras* library with default configuration.

Table 1. Clustering Algorithm Comparisons

Pipeline	Clustering Method	Outliers	Recovered Accuracy
Control - No Clustering / Outlier Detection	N/A	0	18.7425
Clustering Only	K-means	0	30.97082
	DBSCAN	0	20.25868
	Distribution	0	20.07547
Outlier Detection Before Clustering	K-means	14301	47.43767
	DBSCAN	14301	37.87043
	Distribution	14301	38.00366
Outlier Detection After Clustering	K-means	28602	38.10119
	DBSCAN	14301	38.60343
	Distribution	28602	38.00964
Outlier Detection Before & After Clustering	K-means	19294	52.53579
	DBSCAN	23050	48.80418
	Distribution	14738	38.64695
Clustering + Accuracy Tuning	K-means	18036	100.0000
	DBSCAN	40104	100.0000
	Distribution	19557	100.0000
Outlier Detection Before & After Clustering + Tuning	K-means	28064	100.0000
	DBSCAN	39921	100.0000
	Distribution	26273	100.0000

Dataset. The data we used for experimentation is a healthcare dataset [10] containing *age, gender, height, weight, ap_ hi, ap_ lo, cholesterol, gluc, smoke, alco, active*, and *cardio*. The columns are broken down into two types. *age, height, weight, ap_ hi,* and *ap_ lo* are continuous and the rest are categorical. The categorical data has two or three options. The table contains 70,000 rows.

3.2 Experiment 1: Cluster and Outlier Detection Impact

For each combination of continuous variables (*age, height, weight, ap_ hi, ap_ lo*) predicting all other fields, we ran every combination of outlier detection, clustering, and pipeline workflow and output both the number of outliers (representative of the storage costs of the outlier table) and the recovered accuracy. The recovered accuracy is the percentage of the recovered data that is within a guaranteed error threshold (we used 5%) of the original data. Table 1 shows the outcome for height predicting weight.

We compared the results of our postdiction pipeline to the control, which used only machine learning to represent original postdiction with same accuracy of 5%. In all cases, outlier detection and clustering improved the accuracy of original postdiction, though these accuracy benefits must be weighed against the additional storage costs. Thus, the aim was to reduce these additional storage costs by storing the minimal number of outliers while maintaining a high recovered accuracy.

Outlier Detection Algorithm Comparison. We used the default settings for both outlier detection methods and found that they both performed similarly well. In their default settings, Isolation Forest's lower threshold causes it to store more outliers than Z-score, but both methods allow for adjustments using their hyperparameters. We used Z-score outlier detection as the basis of comparison moving forward because of its straight-forward and reproducible methodology and its lower outlier storage costs at the default values.

Clustering Algorithm Comparison. We found that K-means and GMM performed the best on our dataset, with 36% of columns performing best using a K-means clustering algorithm, and 72% of columns performing best using GMM (including ties, which bring the sum to greater than 100%). The differences among columns indicate that there is no ideal single clustering algorithm applicable for the entire dataset, and rather each column must be evaluated to determine the most suitable clustering algorithm.

We concluded that DBSCAN is not an effective algorithm for our dataset because DBSCAN removes points that it considers outliers from the cluster/clusters and saves them as their own cluster. For our dataset, DBSCAN's optimal value of epsilon often leads to one very poor cluster containing only outliers. Therefore, we had to choose a value of epsilon (we used 15) optimized for our dataset to create more meaningful clusters. Even so, DB-Scan only performed best in 4% of the cases, all of which were tied with another algorithm.

Pipeline Comparison. Looking at the control, which had no outlier detection or clustering performed on the data, we observed very low accuracy, but no outliers had to be saved. Conversely, performing outliers detection before training the model and then accuracy tuning after training the model resulted in very high accuracy with many outliers stored. This is another example of the tradeoff between outlier storage and recovered accuracy.

While accuracy tuning can guarantee that 100% of the recovered data satisfies the 5% error threshold, it will always require storing the same or many more outliers as without accuracy tuning. This tradeoff can be further manipulated by setting a higher or lower error tolerance, with a higher error tolerance requiring storing fewer outliers but guaranteeing a lower recovered accuracy.

Finally, we hypothesized that running outlier detection before and after clustering would improve the machine learning models, meaning that the accuracy tuning would need to pick fewer additional outliers. However, we found that the best pipeline configuration for this dataset was to use K-Means Clustering and Accuracy Tuning without Outlier Detection as it had the fewest number of outliers while maintaining the error threshold.

3.3 Experiment 2: Optimizing Multi-Column Storage Performance

While some algorithms/workflows generally performed better, overall the ideal combination of outlier detection, clustering, and workflow was unique to each combination of fields. As mentioned earlier, ideally, we would like to decay the maximum number of columns using the minimum number of features. To that end, we used each continuous variable to predict all the other features in the dataset with every combination of the above variables for each. Then, we chose

Table 2. Minimum Percentage Outliers when Column predicts Row

	age	height	weight	ap_hi	ap_lo
age (16 bits)		**20.33286**	**20.35571**	**20.29857**	**20.34857**
height (16 bits)	**10.45571**		25.76429	**20.40714**	**19.71429**
weight (32 bits)	28.36571	25.76571		37.81571	38.19286
ap_hi (16 bits)	28.26857	36.17571	38.32857		33.55857
ap_lo (16 bits)	31.25143	31.29429	38.18143	31.24857	
cholesterol (2 bits)	25.6028571	25.8171429	25.9528571	26.2242857	26.0557143
gluc (2 bits)	**20.43**	**20.43**	**20.43**	**20.43**	**21.18857**
smoke (1 bit)	**3.131429**	**3.365714**	**3.047143**	**0**	**5.97**
alco(1 bit)	**1.757143**	**1.76**	**1.75**	**0**	**0**
active (1 bit)	**0**	**0**	**0**	**0**	**0**
cardio (1 bit)	**14.37**	**17.43429**	**16.30286**	31.58571	31.54
Total Outliers	28100	33779	33427	33660	35646
Percentage Outliers	40.14%	48.26%	47.75%	48.08%	50.92%

the minimum percentage outliers that can recover 100% of the data and guarantee a 5% error threshold. The minimum percentage outliers are shown in Table 2.

This table shows the percent of data that would have to be saved as the outlier table if that column was used to recover the row based on a 5% error tolerance. For example, looking at the *age* column, it is observed that if *age* was used to recover *height*, then 10.46% of the data would have to be saved as outliers. If *age* were used to recover *weight* 28.37% of the data would need to be saved as outliers, etc. We used bold text to highlight any value we considered promising which stored under a quarter of the original data as outliers.

Next, we calculated the storage saved using two predictors: *age* and *ap_hi*. We used *age* to predict the bold rows (*height, gluc, smoke, alco, active*, and *cardio*) because *age* had the lowest percentage outliers at 40%. We chose *ap_hi* because, while it had a higher percentage of outliers, it predicted two different 16 bit integer fields (*age* and *height*) rather than just one. Finally, we considered whether using *ap_hi* to predict only *age* and *height*, rather than the additional low-bit fields, would save memory by reducing the number of outliers.

We used the size of the fields to calculate the table storage costs. The original table had 70,000 rows, with each row needing 120 bits to store, so the table cost 1,050 kilobytes in storage. To calculate the storage costs of the postdiction output tables, we calculated the costs to store the outliers table plus the recovery table of non-outliers, stored without the columns which can be reconstructed using the saved machine learning models. Overall, the total storage size was:

$$outlier\_table\_size = 120 \times \ num\_outliers$$
$$recovery\_table = (120 - recoverable\_column\_size) \times (70,000 - num\_outliers)$$
$$total\_size = \ outlier\_table\_size + \ recovery\_table$$

Postdiction using *age* found 28,100 outliers, and the outlier and recovery tables were 89.03% the size of the original table. Postdiction using *ap_hi* found 33,660 outliers, and the outlier and recovery tables were 83.99% the size of the original table. Finally, postdiction using *ap_hi* to estimate only *age* and *height* found 24,290 outliers, with the outlier and recovery tables at 82.59% the size of the original table. Using *ap_hi* to predict only *age* and *height* therefore performed

Table 3. Data storage size with and without postdiction

Size/Dataset (KB)	Healthcare Dataset	Healthcare Dataset × 10	Healthcare Dataset × 20
Original Table Size	1050	10500	21000
Outlier Table Size	364.35	5081.025	10197.96
Recovery Table Size	502.81	3973.915	7921.496
Total Size	867.16	9054.94	18119.456
Size with Models	1779.16	9966.94	19031.456

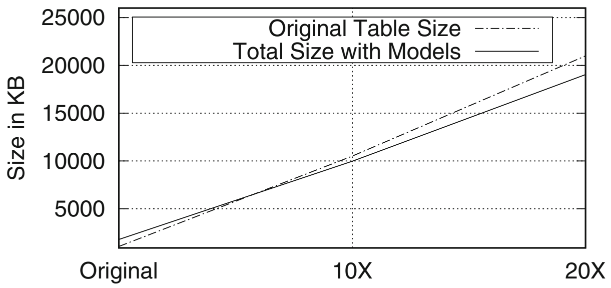


Fig. 4. Data storage size with and without the postdiction pipeline on the healthcare dataset, the dataset copied 10 times, and the dataset copied 20 times

the best. In this case it is better to predict only high-value fields like integers in order to keep outliers low, rather than predicting many binary fields which do not lead to many storage savings in the recovery table.

The above calculations do not take into account the cost of storing the trained LSTM models. LSTMs are robust models with significant storage costs (each model is 228 KB), and a model is created not just for each recoverable field, but for each cluster. Using *ap_hi* to recover age and height requires four LSTM models, totaling 912 KB. Assuming the proportion of outliers stays the same and postdiction can reduce the output table size by 17.41%, the original table would have to be approximately 5,238 KB in order for storage benefits to outweigh the costs of the LSTM. This clearly indicates postdicton's applicability to files of certain size and above (large files/big data). This is also shown in Table 3 where postdiction saved 533.06 KB or 5% on the original dataset expanded 10 times (10,500 KB) and 1968.544 KB or 10% on the original dataset expanded 20 times (21,000 KB). It should be noted that the proportion of outliers remains relatively constant when the dataset is doubled from 10,500 to 21,000 KB and after the crossover point in Fig. 4, the gap between the original size and the total size produced by the postdiction pipeline increases with the size of the dataset, reflecting the gains in savings.

Finally, we can adjust the storage-accuracy tradeoff by changing the error threshold. Selecting an error threshold of 5%, the default of this paper, yielded 24,296 outliers. A higher error threshold yields significant benefits; 10% reduces the number of outliers over 80% to 4,447 outliers. At an error threshold of around 20% there are little benefits for further tolerance, as the number of outliers is now less than 0.1% of the original database.

4 Conclusion

Postdiction is a powerful new concept that gives users the ability to have more control over the storage of their data. In this paper, we proposed the first postdiction pipeline utilizing clustering, outlier detection, and accuracy tuning to increase the accuracy of the original postdiction. We showed how this pipeline could be applied to decay a healthcare dataset and demonstrated the potential that at a large scale, the pipeline will create storage savings.

In this paper, we have only illustrated the potential of our proposed postdiction pipeline and its capabilities as a general postdiction framework. Additional research could observe how our pipeline functions with different and larger amounts of data, including sequential data. Further experimentation could involve different machine-learning, outlier detection, and clustering algorithms, or implement more complex multi-column table reduction using multiple attributes to predict another, or chaining predicted columns together to increase the number of deletable columns. Finally, data postdiction could be combined with traditional methods of lossless compression to increase storage benefits, though this introduces a new trade-off between storage and the inconvenience of decompression for single-value queries.

References

1. Cormode, G., Garofalakis, M., Haas, P.J., Jermaine, C.: Synopses for massive data: samples, histograms, wavelets, sketches. Found. Trends Databases **4**(1–3), 1–294 (2012)
2. Costa, C., Charalampous, A., Konstantinidis, A., Zeinalipour-Yazti, D., Mokbel, M.F.: TBD-DP: telco big data visual analytics with data postdiction. In: 19th IEEE International Conference on Mobile Data Management, pp. 280–281 (2018)
3. Costa, C., Konstantinidis, A., Charalampous, A., Zeinalipour-Yazti, D., Mokbel, M.F.: Continuous decaying of telco big data with data postdiction. GeoInformatica **23**(4), 533–557 (2019)
4. Kersten, M.L.: Big data space fungus. In: CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research (2015)
5. Kersten, M.L., Sidirourgos, L.: A database system with amnesia. In: CIDR (2017)
6. NIH: All of us research program. https://allofus.nih.gov/
7. Reddy, G.T., et al.: Analysis of dimensionality reduction techniques on big data. IEEE Access **8**, 54776–54788 (2020)
8. Song, J., Guo, C., Wang, Z., Zhang, Y., Yu, G., Pierson, J.M.: HaoLap: A Hadoop based OLAP system for big data. J. Syst. Softw. **102**, 167–181 (2015)
9. Tian, J., et al.: CuSZ: an efficient GPU-based error-bounded lossy compression framework for scientific data. In: ACM International Conference on Parallel Architectures and Compilation Techniques, pp. 3–15 (2020)
10. Ulianova, S.: Cardiovascular disease dataset (2019). https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset
11. Yan, Y., Chen, L.J., Zhang, Z.: Error-bounded sampling for analytics on big sparse data. Proc. VLDB Endow. **7**(13), 1508–1519 (2014)
12. Zhang, R.: Data reduction. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1–6. Springer, Cham (2016)