Sanitizing Clinical Data Using Big Data Frameworks

Hongzhu Jiang[1]

[1]**ShanghaiTech University, Shanghai, China**

Abstract

This project explores privacy-preserving techniques, specifically k-anonymity and l-diversity, for clinical data using distributed computing frameworks like Apache Spark. Due to the siloed nature of healthcare data across institutions, federated k-anonymity is leveraged to enable collaborative data analysis without direct data sharing. The distributed implementation follows the methodology outlined in a previous study on the TDS algorithm, which provides a scalable approach for implementing k-anonymity in a distributed environment. This project aims to develop a scalable, privacy-preserving framework that balances privacy protection and data utility while ensuring compliance with privacy regulations such as HIPAA and GDPR.

Introduction

In the era of big medical data, datasets often contain quasi-identifiers—combinations of non-sensitive attributes like age, gender, and ZIP code—that, when combined, can uniquely identify individuals. Even in anonymized datasets, these quasi-identifiers can lead to privacy breaches when adversaries link them with external data sources. Sensitive attributes such as income need to be explicitly protected to prevent unauthorized disclosure.

A widely adopted solution to mitigate these risks is k-anonymity, which ensures that each individual in the dataset is indistinguishable from at least *k-1* others based on their quasi-identifiers. However, k-anonymity has limitations. For example, if all individuals in a k-anonymous group share the same sensitive attribute value, an adversary can still infer that sensitive value with certainty, which exposes the data to attribute disclosure risks. To address this, l-diversity extends k-anonymity by requiring each equivalence class to contain at least *l* distinct sensitive values, reducing the risk of exact inference. However, this approach does not fully mitigate the possibility of probabilistic inference, especially when one sensitive value dominates within the group.

While these privacy models offer protections, enforcing stronger privacy guarantees often comes at the expense of data utility and computational efficiency, particularly with large-scale clinical datasets such as electronic health records (EHRs). As a result, there is a pressing need for scalable, distributed privacy-preserving algorithms that can balance privacy and utility, while supporting regulatory compliance (e.g., HIPAA, GDPR).

A significant challenge in healthcare data privacy arises from the fact that data is often siloed across multiple institutions and cannot be directly shared due to privacy concerns and regulatory restrictions. In response, federated k-anonymity offers a promising solution. By utilizing distributed computing, it enables institutions to collaboratively process sensitive data without sharing it directly. This approach not only addresses privacy concerns but also facilitates secure, large-scale data analysis across multiple institutions. The distributed implementation of k-anonymity in this project follows the methodology of the TDS algorithm—a distributed approach for scalable k-anonymity—ensuring both privacy protection and computational efficiency.

This project aims to implement and evaluate k-anonymity and l-diversity techniques on clinical data using Apache Spark, with a particular emphasis on scalability for real-world datasets. The distributed k-anonymity implementation is inspired by the TDS algorithm, aiming to balance privacy protection, computational efficiency, and regulatory compliance.

Background

The protection of sensitive information in large datasets has been a long-standing challenge, particularly in the era of big data. One of the most widely adopted privacy-preserving models is k-anonymity, introduced by Sweeney (2002)[1]. In this model, data is anonymized such that each individual cannot be distinguished from at least $k-1$ others based on quasi-identifiers such as age, gender, or ZIP code. This ensures that the privacy of individuals is preserved, even in datasets that may otherwise be vulnerable to re-identification. K-anonymity has become a foundational concept in privacy research, especially in fields like healthcare, where the confidentiality of personal data is crucial.

Building upon k-anonymity, LeFevre (2006)[2] proposed Mondrian multidimensional k-anonymity, an extension designed to handle complex, multidimensional datasets. This approach divides the data into regions and ensures that each region satisfies the k-anonymity requirement. Mondrian k-anonymity provides greater flexibility for anonymizing datasets that have multiple sensitive attributes, allowing for the preservation of privacy while improving the data's utility in certain contexts. This technique is particularly useful in scenarios where datasets contain various attributes that must be anonymized across multiple dimensions.

While k-anonymity and its extensions, such as Mondrian, provide strong privacy protection, they are still vulnerable to certain types of privacy attacks. Machanavajjhala et al. (2007)[3]introduced l-diversity, an enhancement that aims to address these limitations. In l-diversity, each equivalence class must contain at least *l* distinct sensitive values, preventing attackers from inferring sensitive information even if they are able to identify the equivalence class. This addition to the k-anonymity model ensures that the anonymized data is more resilient to attribute disclosure risks, further improving privacy protection.

Following this, Li (2007)[4] introduced t-closeness, a model that goes beyond k-anonymity and l-diversity by ensuring that the distribution of sensitive attributes in each equivalence class is close to the overall distribution in the dataset. This measure prevents attackers from inferring sensitive information by observing the distribution of attributes in anonymized datasets, thus providing an additional layer of privacy protection.

In the context of biomedical data, Prasser et al. (2014)[5] introduced ARX, a comprehensive tool for anonymizing sensitive data. ARX incorporates various privacy models, including k-anonymity, and is designed specifically for large, sensitive datasets such as those found in healthcare. ARX allows for the application of different anonymization techniques while balancing the trade-offs between privacy protection and data utility. However, ARX operates as a centralized tool, and its scalability becomes an issue when dealing with large, distributed datasets, particularly in settings where data from multiple institutions must be anonymized.

The scalability of traditional anonymization tools like ARX can be limited when handling big data, especially in fields such as healthcare. Single-machine solutions struggle with the computational demands of large datasets, often resulting in inefficiencies and high costs. This is where distributed computing plays a crucial role. By leveraging distributed platforms such as Apache Spark, anonymization processes can be scaled efficiently, allowing for parallel processing of massive datasets across multiple nodes. This approach not only accelerates the anonymization process but also ensures that sensitive data can be anonymized without the need for direct data sharing between institutions, a critical factor in maintaining privacy and complying with regulations such as HIPAA and GDPR. Distributed anonymization also provides significant computational advantages, making it possible to process large-scale datasets in a time-efficient manner.

Related Work

Apache Spark is an open-source, distributed computing framework designed for processing large-scale datasets quickly and efficiently. Unlike traditional data processing frameworks like MapReduce, Spark provides in-memory processing, which significantly accelerates computation by reducing the need to read and write intermediate data to disk. Spark's core abstraction, the Resilient Distributed Dataset (RDD), allows for distributed data processing with fault tolerance. RDDs enable parallel computation and can be split across multiple nodes in a cluster, providing both scalability and speed. Each RDD is immutable, ensuring that transformations on data are applied in a way that doesn't alter the original dataset, thus allowing for efficient data lineage tracking. Spark also supports a wide range of programming languages, including Java, Scala, Python, and R, making it versatile for various use cases.

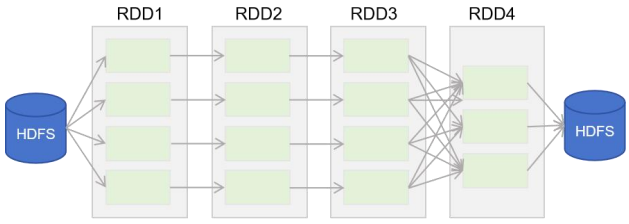

Figure 1. Apache Spark with HDFS Data Distribution.

One of Spark's core advantages is its ability to scale across thousands of nodes in a cluster, allowing it to handle both batch and real-time data processing at an unprecedented scale. Its ability to process complex queries, perform machine learning tasks, and execute data analytics in a distributed manner makes it an ideal choice for implementing privacy-preserving algorithms, such as k-anonymity and l-diversity, on large-scale datasets. Additionally, Spark's ecosystem includes libraries such as Spark SQL[6] for querying structured data, MLlib[7] for machine learning, and GraphX for graph processing, providing a comprehensive platform for diverse analytical needs. These features make Apache Spark a powerful tool for ensuring efficient and scalable privacy-preserving techniques in the healthcare domain.

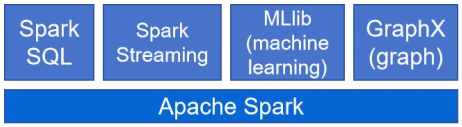

Figure 2. Components of the Apache Spark Ecosystem.

The problem of privacy-preserving data analysis has garnered significant attention in recent years, especially in the context of big data processing. Several studies have explored the application of k-anonymity and other anonymization techniques using distributed frameworks such as Apache Spark.

Sopaoglu and Abul (2017)[6] proposed a top-down k-anonymization approach implemented on Apache Spark for large-scale data. Their approach leverages Spark’s distributed computing capabilities to efficiently process large datasets while ensuring that data privacy is preserved. They focus on the optimization of the top-down specialization technique for achieving k-anonymity, where they use Spark’s parallelism to accelerate the computation process, which is crucial for handling massive datasets in big data environments. The authors also highlight the importance of reducing computational cost while maintaining data privacy, which is a challenge for traditional privacy-preserving methods.

Similarly, Canbay(2017)[7] explored big data anonymization techniques using Apache Spark. They focus on the scalable implementation of anonymization algorithms to process large datasets while ensuring compliance with privacy standards. Their work demonstrates the advantages of using Spark in terms of parallel data processing and efficiency, especially for anonymization tasks on large-scale healthcare and other big data domains. They also emphasize the need for distributed approaches that allow organizations to anonymize their data without compromising on privacy or performance.

Tortikar (2019)[8] proposed an efficient implementation of k-anonymization using Apache Spark to address the challenges of privacy preservation in big data environments. His approach leverages the parallel processing power of Spark to handle large-scale datasets, enabling faster computation of k-anonymity than traditional methods. Tortikar emphasizes the importance of reducing computational time while maintaining privacy standards, especially when working with sensitive data in distributed settings. The work highlights the trade-offs between data utility and privacy, providing a framework that balances both aspects in big data anonymization tasks. The study demonstrates the potential of Apache Spark as a scalable solution for privacy-preserving data analysis, particularly in applications requiring the anonymization of large datasets in healthcare and other sectors.

Methods

Due to the NP-completeness of optimal k-anonymization, researchers have developed a variety of heuristic and suboptimal approaches. This project follows the Top-Down Specialization (TDS) algorithm proposed by Uğur Sopaoglu, with modifications and extensions based on their publicly available implementation. Leveraging the Apache Spark framework, we parallelized the classical TDS algorithm in a distributed computing environment to enable efficient k-anonymity processing. The workflow for Apache Spark based TDS is shown in Figure 3.

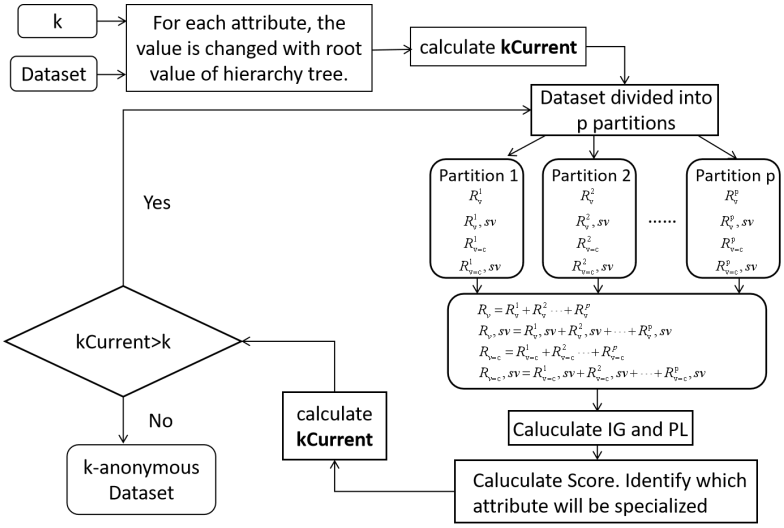

Figure 3. Workflow for Apache Spark based TDS (adapted from Sopaoglu and Abul, 2017).

As with all generalization-based anonymization methods, the TDS algorithm requires a domain hierarchy tree for each quasi-identifier attribute. The algorithm performs a coordinated top-down traversal over these trees, one for each quasi-identifier, in order to determine suitable generalization paths. The TDS algorithm consists of the following main steps:

1. Pre-processing: All non-quasi-identifier attributes are removed from the dataset.
2. Generalization: The algorithm maintains a set of anonymization-level (AL) trees. Initially, each tree corresponds to the full domain hierarchy of a quasi-identifier. In each iteration, the dataset is generalized based on

the Attribute Lattice (AL), meaning that each attribute value in a record is recursively generalized from its leaf node toward the root of its hierarchy tree. If the current anonymization level results in $kCurrent > k$, indicating that the data is over-generalized, further specialization is performed. Otherwise, the algorithm terminates, and the resulting dataset satisfies k-anonymity. The notation used in the algorithm is summarized in Table 1.

Table 1. Notation Used in the Algorithm.

Notation	Description
kCurrent	the calculated *k* value at each iteration of anonymization
p	number of partition
R_v	set of records containing attribute values which can be generalized to v
R_v, sv	set of data records which contains sensitive values sv in R_v
IG	Information Gain
PL	Privacy/anonymity Loss
IGPL	Information Gain Privacy Loss

Experiments

In this project, Apache Spark was deployed in Standalone cluster mode to enable distributed processing of the k-anonymity algorithm. The cluster consisted of one Master node and two Worker nodes, which collaboratively handled task scheduling and parallel computation. The deployment used client mode for job submission, allowing easier debugging and log access.

A structural diagram illustrates the Spark environment deployed under Standalone mode with client-based job submission. In this architecture, the Master node is responsible for resource management and task scheduling, while the Worker nodes perform the actual computations. Each executor occupies 2 CPU cores, so each Worker can host up to two executors concurrently under the current configuration.

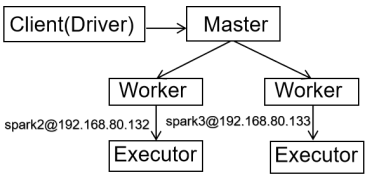

Figure 4. Spark Architecture Diagram (Standalone Cluster Mode).

To improve computational efficiency and memory usage, the following resource configurations were applied: each executor was allocated 2 GB of memory and 2 CPU cores, with a total of 4 executor cores available across the cluster. The driver program was executed on the Master node. Table 2 summarizes the configuration details of the Spark Standalone cluster used in this project. As shown in Figure 5, Spark 4.0.0, Scala 2.13.0, and JDK 17.0.12 were installed under the /opt directory on the Master node to support the cluster runtime environment.

Table 2. Cluster Setup Configuration.

Parameter	Value
Cluster mode	Standalone
Master URL	spark://jhz:7077
Number of worker	2
Spark version	4.0.0
Scala version	2.13.0
Java version	17.0.12

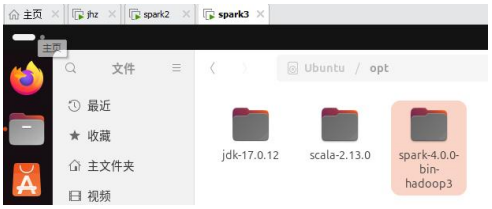

Figure 5. Spark-related Software Installation.

Job submission was performed using the "spark-submit" command, specifying the main class, input file path, k value, and output directory. The Scala project was packaged as an executable JAR file. The overall goal of this configuration was to maximize distributed execution efficiency while ensuring correctness of the anonymization results. As shown in Figure 6, the Spark cluster was successfully launched using the start-all.sh script on the Master node, initiating both the Master and Worker processes. Figure 7 shows the screenshot of the Spark UI is provided below, showing the resource allocation after the cluster is launched.

```
jhz@jhz-VMware-Virtual-Platform:/opt/spark-4.0.0-bin-hadoop3/sbin$ ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark-4.0.0-bin-hadoop3/logs/spark-jhz
-org.apache.spark.deploy.master.Master-1-jhz-VMware-Virtual-Platform.out
spark2@192.168.80.132: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark-4.0.0-bin
-hadoop3/logs/spark-spark2-org.apache.spark.deploy.worker.Worker-1-spark2-VMware-Virtual-Platform.out
spark3@192.168.80.133: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark-4.0.0-bin
-hadoop3/logs/spark-spark3-org.apache.spark.deploy.worker.Worker-1-spark3-VMware-Virtual-Platform.out
```

Figure 6. Cluster startup via start-all.sh on the Master node.

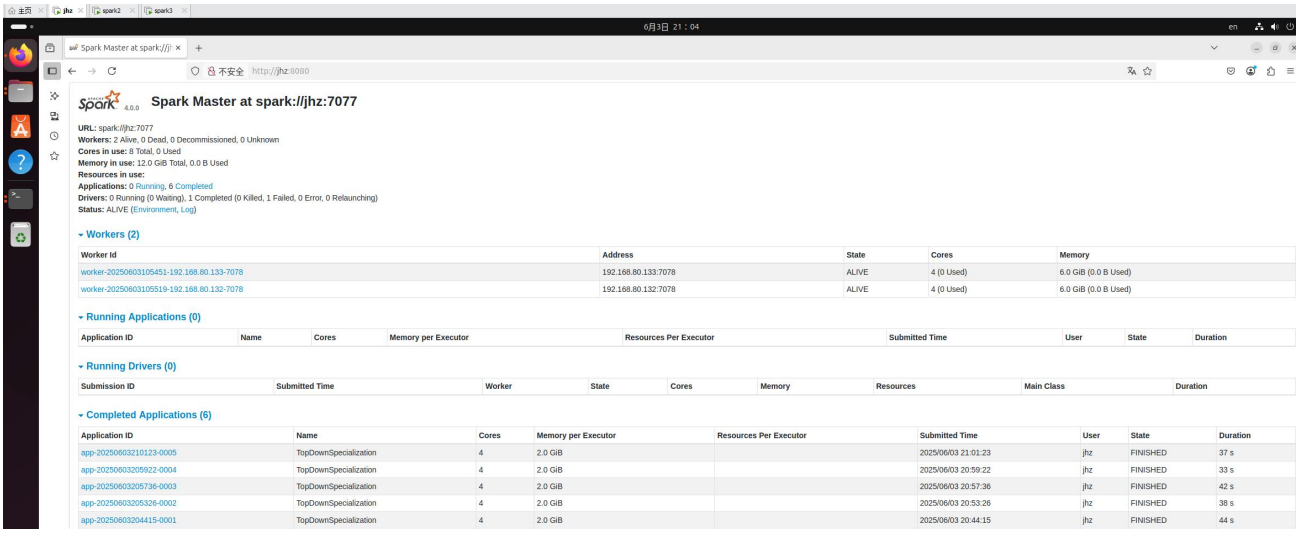

Figure 7. Spark Web Monitoring Page.

Two datasets were used in this study to evaluate the performance of the proposed k-anonymity algorithm in different scenarios.

(1) SyntheticMass Dataset:

The first dataset was obtained from the SyntheticMass project, which provides publicly available synthetic health records. This subset contains 116 records, each including attributes such as BIRTHDATE, gender, race, and income. The age attribute was derived from the BIRTHDATE field through preprocessing. The set of quasi-identifiers used for anonymization is: {gender, age, race}, and the sensitive attribute is income.

To enable generalization-based k-anonymization on the test1_data dataset, we manually constructed taxonomy trees (hierarchies) for each quasi-identifier attribute: gender, race, and age. These hierarchies define the permissible generalization paths and serve as the basis for the Top-Down Specialization (TDS) algorithm. Figure 8 illustrates the designed hierarchies, where each node represents a generalization level, and parent-child relationships reflect abstraction levels.

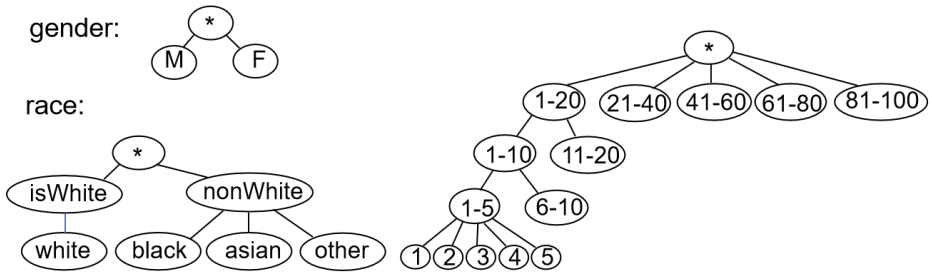

Figure 8. Gender, Race and Age Hierarchies.

(2) Adult Dataset (adult.csv)

The second dataset is the widely used Adult dataset from the UCI Machine Learning Repository, containing 32,561 records. It includes the following quasi-identifiers: {sex, race, relationship, workclass, native_country, occupation, education, marital_status}, and the sensitive attribute remains income. This dataset presents a more complex anonymization challenge due to the higher dimensionality of quasi-identifiers and the larger sample size.

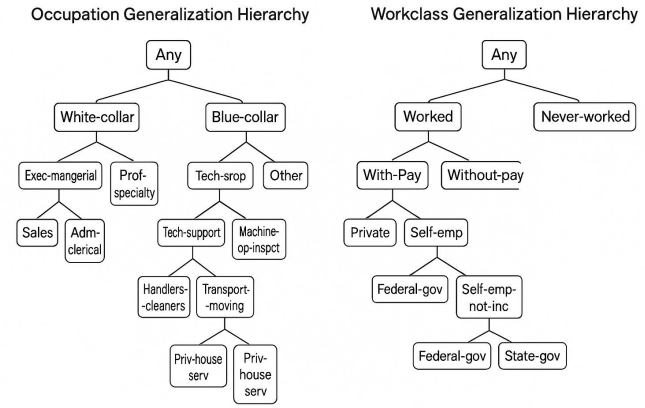

Figure 9. Occupation and Workclass Hierarchies.

Results

The data utility comparison between the distributed TDS implementation and the local ARX software was conducted on the SyntheticMass dataset under the condition of k=3. Using the Normalized Certainty Penalty (NCP) as the evaluation metric[11], the ARX-anonymized dataset achieved a lower overall NCP of 0.5833, whereas the TDS-anonymized dataset exhibited a higher NCP of 0.7274, as shown in Figure 10. This result suggests that ARX preserved data utility more effectively by applying less aggressive generalization, while the TDS approach introduced broader generalizations, leading to increased information loss.

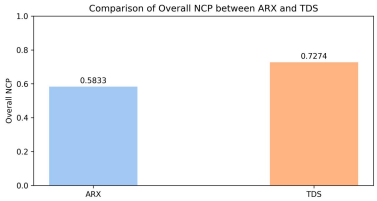

Figure 10. Comparison of Overall Normalized Certainty Penalty (NCP) between ARX and TDS Methods.

To further investigate the source of utility loss, Figure 11 presents a detailed attribute-wise comparison between the two methods. ARX incurred the highest penalty in the age attribute, likely due to its finer-grained numeric domain requiring broader generalization to satisfy k-anonymity[12]. In contrast, TDS preserved more detail for age but applied stronger generalization to gender and race, resulting in higher NCP values for those categorical attributes. This suggests that TDS's specialization strategy favors preserving utility on attributes with higher domain cardinality, while tolerating greater loss on simpler categorical attributes.

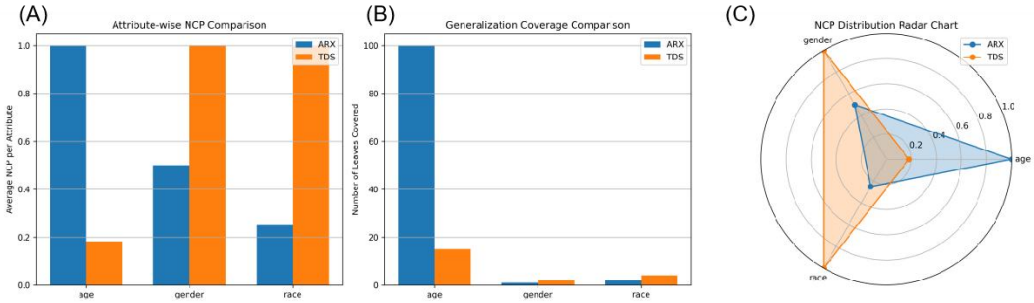

Figure 11. Comparison of NCP Metrics between ARX and TDS Methods. **(A)** Average NCP per attribute for ARX and TDS. **(B)** Generalization Coverage Level for Each Attribute. **(C)** Radar chart showing attribute-wise NCP distribution comparison.

To evaluate the performance of the TDS algorithm, I measured the running time required to achieve k-anonymity across two dataset. For each k value tested, I ran the anonymization process twice and reported the average time. As shown in the Figure 12 and Figure 13, the running time decreases as k increases. This trend holds across both datasets but is more pronounced in the larger dataset. The reason is that larger k values allow the algorithm to satisfy the anonymity requirement at higher levels of the generalization hierarchy, reducing the need for deeper, more granular refinements. In contrast, smaller k values require more fine-grained generalizations to ensure that all equivalence classes meet the size threshold, resulting in increased computation time.

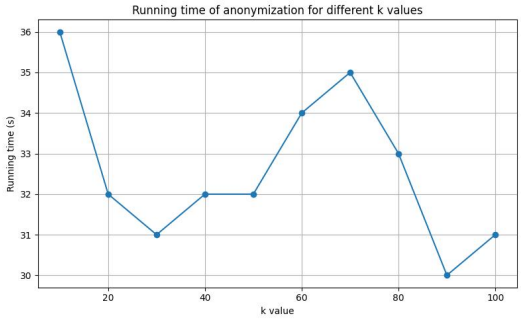

Figure 12. Efficiency results on the SyntheticMass Dataset (116 Records).

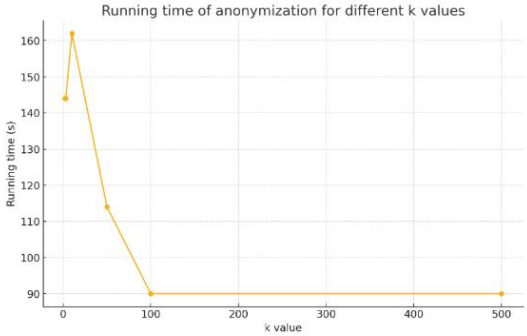

Figure 13. Efficiency results on the Adult Dataset (32,561 Records).

Discussion

The experimental results demonstrate distinct trade-offs between the ARX and distributed TDS anonymization approaches in terms of data utility and performance. In the utility evaluation based on the Normalized Certainty Penalty (NCP), ARX produced lower overall NCP scores compared to TDS. This indicates that ARX is capable of achieving k-anonymity while preserving more data specificity, especially for categorical attributes such as gender and race.

In terms of performance, the timing results indicate that the computational cost of the TDS algorithm is inversely related to the k value. Higher k values allow the algorithm to generalize at higher levels of the taxonomy tree, leading to fewer iterations and shorter running time. This trend was more evident in the larger dataset, suggesting that the computational benefits of coarser generalization scale more effectively with data size. Nevertheless, the performance evaluation also highlights a potential limitation of the TDS approach: under stricter privacy requirements (i.e., lower k), the algorithm incurs substantially higher computational overhead, which may challenge its efficiency in latency-sensitive or resource-constrained environments.

A notable limitation of this study lies in the scale and diversity of the experimental dataset. The generalization behavior and performance observed here may not fully generalize to more complex healthcare or transactional datasets with millions of records. To address this, future work should include large-scale real-world datasets to rigorously evaluate how anonymization strategies perform in production-level environments and under diverse attribute distributions.

Conclusions

This study compared the data utility and performance of a distributed Top-Down Specialization (TDS) implementation with the widely used local ARX software under a shared k-anonymity constraint. Experimental results on the SyntheticMass dataset show that ARX consistently achieves better data utility, as reflected in its lower overall and attribute-wise Normalized Certainty Penalty (NCP) scores. This can be attributed to ARX's ability to

apply more fine-grained generalization strategies in a centralized setting. In contrast, the TDS approach, while introducing greater information loss, demonstrates better scalability and is particularly advantageous in distributed environments where data cannot be centrally aggregated due to privacy, regulatory, or organizational constraints. By enabling collaborative anonymization without requiring raw data sharing between institutions, TDS provides a practical and privacy-preserving solution for multi-party data publishing.

References

1. Sweeney, Latanya. "k-anonymity: A model for protecting privacy." International journal of uncertainty, fuzziness and knowledge-based systems 10.05 (2002): 557-570.
2. LeFevre, Kristen, David J. DeWitt, and Raghu Ramakrishnan. "Mondrian multidimensional k-anonymity." 22nd International conference on data engineering (ICDE'06). IEEE, 2006.
3. Machanavajjhala, Ashwin, et al. "l-diversity: Privacy beyond k-anonymity." Acm transactions on knowledge discovery from data (tkdd) 1.1 (2007): 3-es.
4. Li, Ninghui, Tiancheng Li, and Suresh Venkatasubramanian. "t-closeness: Privacy beyond k-anonymity and l-diversity." 2007 IEEE 23rd international conference on data engineering. IEEE, 2006.
5. Prasser, Fabian, et al. "Arx-a comprehensive tool for anonymizing biomedical data." AMIA Annual Symposium Proceedings. Vol. 2014.
6. Armbrust, Michael, et al. "Spark sql: Relational data processing in spark." Proceedings of the 2015 ACM SIGMOD international conference on management of data. 2015.
7. Meng, Xiangrui, et al. "Mllib: Machine learning in apache spark." Journal of Machine Learning Research 17.34 (2016): 1-7.
8. Sopaoglu, Ugur, and Osman Abul. "A top-down k-anonymization implementation for apache spark." 2017 IEEE international conference on big data (big data). IEEE, 2017.
9. Canbay, Yavuz, and Seref Sağıroğlu. "Big data anonymization with spark." 2017 International Conference on Computer Science and Engineering (UBMK). IEEE, 2017.
10. Tortikar, Pratik. "K-Anonymization implementation using apache spark." (2019).
11. Kim, Soonseok. "Optimizing Privacy in Set-Valued Data: Comparing Certainty Penalty and Information Gain." Electronics 13.23 (2024): 4842.
12. Onesimu, J. Andrew, et al. "Privacy preserving attribute-focused anonymization scheme for healthcare data publishing." IEEE Access 10 (2022): 86979-86997.