Politecnico di Torino

Politecnico di Torino

Master's Degree in Mathematical Engineering - Data Analysis
A.a. 2024/2025
Graduation Session July 2025

Generation of Synthetic Tabular Data for Controlled Machine Unlearning

Supervisors:
Professor Giobergia Flavio
Savelli Claudio

Candidate:
Gianusso Fabio

Abstract

One of the most recent areas of research in the field of Artificial Intelligence concerns Machine Unlearning. Since research necessarily needs many tests and benchmarks to evaluate new proposals about methods or algorithms and compare them to the already existing ones, large datasets are requested.

Complications about the difficulty to collect big amounts of data or privacy of individuals led to synthetic datasets. Not only they are faster to create than collecting real data, and don't leak information about anyone, but they are also built ad hoc, so the user can decide,and tune characteristics based on the aim of the test.

In this thesis both Machine Unlearning and Synthetic Data Generation topics will be analysed, and the focus will be about showing how with different synthetic datasets tuned on purpose, Machine Unlearning can become harder or easier, and how we can use these synthetic datasets to observe performances of Machine Unlearning methods and compare them.

The core idea is that samples from more entangled datasets will be harder to be forgotten by the models, while datasets with more separated data will be easier. We will also observe what having entangled data means and how we can measure this phenomenon.

Summary

1. Introduction

Recently, Machine Learning has become very popular in different applications not only of our daily life but also in many specialized areas of any scientific field. Surely this technological innovation has brought notable advantages and has facilitated various tasks, but it also raised some concerns. First, finding data is not so easy as one could think, and Machine Learning algorithms often require large amount of data to be trained properly and be efficient, then problems about privacy and data security are increasingly emerging, especially when it comes to sensitive information of private citizens.

The concept is that models could use indirectly or not information about individuals for their predictions, also without their permission. Nowadays this is considered a privacy violation.

Since General Data Protection Regulation was introduced, every user got the right to request their data to be deleted from databases and models. Retraining models every time a request is made would be impossible, as well as extremely expensive in terms of energy, time and money.

Beyond that, due to the very large amount of data necessary to train big models, often developers could realize that some data were wrong or biased, or subjected to copyright, and retraining from scratch the entire model it would essentially mean having wasted all the work done up to that point.

Indeed, this would me an extremely expensive operation and repeating it every time that is necessary for the model to forget some information would be unrealizable.

From these and other needs, Machine Unlearning was born.

1. Machine Unlearning

So, the question is: "*Is it possible to remove the effects of some samples from a model trained on these, preserving its effectiveness, without retraining it from scratch on a new training set?*"

The answer is that this is exactly what the Machine Unlearning is supposed to do.

Machine Unlearning indeed consists in eliminating the influence of a portion of data from a trained model, without requiring a full retraining on another subset of the original dataset, which, as said, is often a very expensive process.

Formally, we have a training set T, a forget set F, that is a subset of the entire training set that contains samples to forget, and a retain set $R = T / F$, that contains samples that should remain in the training of the Unlearned model.

The aim is to obtain, after the Unlearning process on the Full model, trained on T, a model that is the closer to the Gold model, that is a model trained from scratch on R.

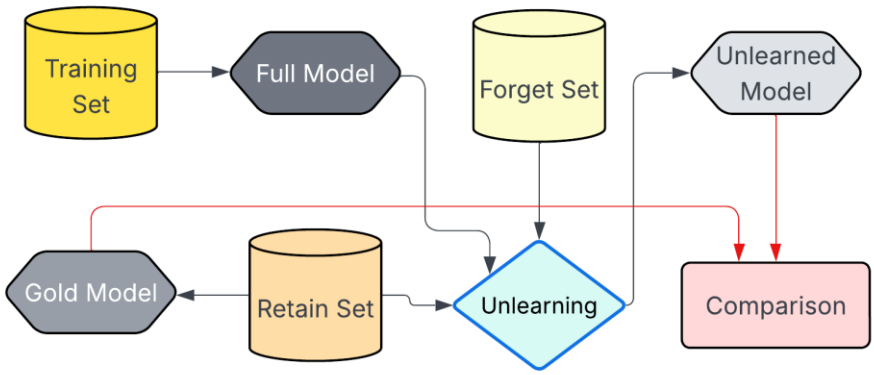

Figure 1 Machine Unlearning Framework

Evaluating the Machine Unlearning methods is a significant challenge, and the objective is to compare the Unlearned Model and the Gold Model, since the first is wanted to be the closer to the Gold, that having being trained on the retain set that don't contains the forget samples, is exactly what we should require as a result from the unlearning process..

Since one of the main reasons for removing data from trained models is about privacy, the aim is forgetting *identities*, that are the set of samples related to a specific individual. However, the model, after the unlearning process, has still to be efficient and produce accurate results. Thus, the comparison between Unlearned Model and Gold model is not only about performance, as we want that even without the forgotten data the predictions are reliable, but also about how much those data have actually been forgotten.

To quantify the goodness of removing data-related information, a Membership Inference Attack (MIA) was made on models. This attack consists in taking features from predictions of a model for two datasets, in our case the set of the data to be forgotten (Forget Set) and a set of unseen data (Test Set), assigning them a label that indicates from which set a sample is, and then train a binary classification model (via Logistic Regression) to detect how easy is to distinguish if the model has seen a sample or not observing its prediction for it.

Examples of features from the predictions that could be taken are *logits*, entropy, probabilities (for each class), maximum confidence and correctness of prediction (as binary variable). As will be mentioned, we also tried to use the outputs of the model before the last layer as features.

Logits are scores that the model gives as result for a sample to classify. Usually, to classify it we take the highest and assign as prediction the class with the highest score. The idea here is that observing how confident is the model in classifying a sample, the MIA could divide unseen data from seen-but-forgotten ones. The lower is the difference of MIA results, the better was the unlearning process, and the higher is the guaranteed privacy, because it means that the Unlearned Model act very similar to the Gold one, that actually has never seen both given sets, when it comes to classify a sample, even if it was seen in training before the Unlearning. We should expect a MIA accuracy result near 0.5, that means random guess, for the Gold model, and a higher one for the Full model (trained on the entire training set). The Unlearned model will have a MIA score between those value, the closer is to the Gold one, the better was the Unlearning. In our experiments we used the Forgetting Score to evaluate the guaranteed privacy by a model, that is the distance between the accuracy of MIA and 0.5, that is the level of perfect privacy guaranteed, as this means that an intruder can only go with random guess to understand if a sample was already seen by a model or not.

2. Synthetic Data Generation

As mentioned above, recent Deep Learning models often require very large amounts of data to be trained, and often benchmark dataset are used to compare or test models. Since collecting data is not easy, and aforementioned concerns like privacy or copyright prevent the use of many resources, Synthetic Data are a reliable solution to access easily large amount of data with relatively little effort in terms of time and money.

Synthetic Data can be generated from scratch or as an augmentation of real datasets.

Main objectives of Synthetic Data generation are:

• Class Balancing: rebalance real datasets in which some classes or groups (defined with respect to specific attributes) are underrepresented.

• Sensitive Information Hiding: to hide datasets in which sensitive information is leaked, creating a new dataset with same characteristics of real one with fictitious individuals.

• Cheap Data: often some types of data require many resources to be collected, tools to generate them could solve this problem and avoid waste of time and money.

When Synthetic Data are generated from a starting dataset for privacy reasons, the aim is to have a new dataset that performs as close as possible as the original one when used for training.

When Synthetic Data are generated for class balancing instead, the problem is that a dataset with some classes or group that are underrepresented could lead to inaccurate predictions on those samples, or biased predictors.

There are two ways to solve this problem:

• Undersampling: we discard some samples from majority classes to obtain an equal number of samples for each group/class.

• Oversampling: we generate new synthetic samples for minority classes/groups until its dimension reaches the majority class one.

Obviously, the field of Synthetic Data is about Oversampling, since the concept is to generate new data. As will be discussed in the next session, many Synthetic Data Generation methods turned out to work better than simple Undersampling.

Often these two techniques are also combined, generating more sample than necessary and then undersampling them to keep the ones that reflects better desired characteristics.

The purpose of this work is to generate synthetic data, in particular tabular data, to test these methods, controlling some precise features of the datasets, and exploring which are their characteristics that make more or less complex the Machine Unlearning process.

2. Related Works

For this thesis we had to collect information both about Synthetic Data Generation strategies and Machine Unlearning methods, and try to adapt them to our case.

1. Synthetic Data Generation

Most of Synthetic Data Generation methods are based on starting from a dataset that already exists, we will see some of them. The main reasons for generating synthetic data are imbalanced datasets or protecting some sensitive information. Both these motivations are about a pre-existing dataset, but in our case, since our objective is to be able to fully control the characteristics of the dataset, we cannot begin from real data, thus we must create a dataset from scratch.

To compare different Synthetic Data generative methods, we can use two types of metrics:

• Utility Metrics: indicate how good is the classification trained on the synthetic dataset. (For example, Accuracy, ROC–AUC)

• Fairness Metrics: indicate how much difference there is in measurements in different subgroups. (For example, Equalized Odds, Statistical Parity, Equal Opportunity)

We can observe briefly what these metrics indicate:

• Accuracy: indicates the proportion of samples correctly classified on the entire dataset. It can be also computed for each class.

• Recall: indicates proportion of samples correctly classified in a class with respect to all samples assigned to that class by the model. The idea is that shows how much del model is sensitive to a class.

• F1 Score: combines accuracy and recall, and indicates the goodness of the performance, it takes values between 0 and 1 and the higher it is, the better is the prediction.

• ROC-AUC: Receiver Operating Characteristics (ROC) curves are created varying a threshold for the classifier and keeping track of True Positive Rate (TPR) and False Positive rate (FPR) and plotting them in a bidimensional graph (restricted to the square [0,1] x [0,1]). Then the Area Under Curve (AUC) is computed, and it indicates the probability that the model will assign a higher score (of belonging to positive class) to a sample that is truly positive than to a negative one. Obviously the higher is AUC, the better is the model.

• Equalized Odds: indicates difference between true positive rates and false positive rates for different subsets of the dataset.

• Statistical Parity: indicates whether the probability of being classificated in a class is consistent across different subsets of the dataset.

• Equal Opportunity: indicates difference between true positive rates for different subsets of the dataset.

Often these metrics must be analysed together, for example a binary model that classify everything as positive will have a perfect accuracy on the positive class, but for sure it is not a good model and only observing, for example, recall we can notice it.

True Positive (TP) are samples classificated as positive (class 1 in a binary classification) that are truly positive, False Positive (FP) instead samples classificated as positive that are negative (misclassified).

Groups are identified as sets of samples with the same value for a specific attribute.

A class-balanced dataset is crucial for data preprocessing in most of classification problem. We define a dataset imbalanced when labels are not distributed homogeneously and there are overrepresented or underrepresented classes. If a model is trained on an imbalanced dataset, it could easily underestimate error on samples from minority classes and also if the overall accuracy is satisfying the performances will be reliable only for some classes.

To overcome this problem one of the most popular and intuitive strategy is undersampling, that consist in reducing the dataset cutting off some samples from majority classes until their cardinality matches the minority ones. Surely this solves the problem of imbalance, but the loss of information due to having less samples in training set often leads to weaker models with a low learning capability.

Another way to tackle this problem is oversampling, that consist in generating new data for minority class until they reach dimensions of majority one. The advantage with respect to undersampling is that no information from the original dataset is lost, but it must be paid attention on how new data are generated.

Generating synthetic data is a powerful tool to adjust imbalanced datasets or to hide sensitive information, but the risk is to increase chances of overfitting (if new samples are too much similar to the already existing ones, they will impact too much on model weights) and to introduce noise into the dataset (if they are distributed too much differently with respect to the original samples).

The balance of a dataset can be not only about classes, but also about some attributes called *protected* that we could need to be uniformly spread among samples, for example even if the label of a dataset is *height*, and *gender* is just a feature, we could need a dataset with the same number of male and females, to train a robust model with a good prediction capability.

To evaluate a synthetic dataset goodness, we must use a classifier to train on it and compare the obtained performances with the ones related to original datasets. For synthetic datasets generated from scratch instead, we can evaluate them observing if the required characteristics are satisfied or not.

Let us analyse some of the most popular Synthetic Data generation methods:

- **SDV - GC**

The algorithm receives the starting dataset in tabular form; it is like an SQL database with different tables and references between them. A table without references to other tables is called standalone table.

SDV– GC tries to estimate the distribution of the data in the final table and then generates new synthetic data samples from it.

For every standalone table indeed, the model creates a distribution for every column and a covariance matrix with the covariances between them. To estimate them, SDV–GC uses Gaussian Copula, an algorithm that converts all column distributions (estimated from their respective values in the starting dataset) to standard Gaussian and then finds covariances.

Then each table is aggregated with their respective parent table, iteratively. Gaussian Copula is applied to the "leaf node" tables (if we imagine tables relationships as a tree), we estimate distributions and covariance matrix for each table and add them as new columns in the respective parent table. This is defined as *extended table*. Repeating this procedure for all tables leads to a single final extended table that resumes all dataset information. Then to create synthetic data we just have to sample from the specified distributions for each column of each row and then obtain respective *children* rows for *children* tables. This procedure can be repeated until enough synthetic data are generated.

- **CART**

As the previous algorithm, also CART creates synthetic data starting from a pre-existing dataset. CART uses classification trees to partition the predictor space and create subsets with almost only samples with the same label. Partitions are created with several binary splits and the leaves of the

tree are the subsets. The tree can be pruned to avoid overfitting, using ad-hoc criteria.

Using tree leads to advantages like capturing also non-linear relationships and interaction effects better than parametric models, but it is also more difficult to interpretate and close to decision boundaries, predictions could be irregular. Intuitively we could say that the algorithm consists in sampling a value for an attribute for an instance from the pool composed by the values of the other instances that are "similar" (that means that the classification tree put them in the same leaf node, and they are in the same region of the partition of the latent space). If we need to synthetize more than a single attribute, we can do it sequentially using every time the last dataset created. To sample from a pool of discrete values (or classes) CART uses the Bayesian Bootstrap, and if the values are continuous, a density function is estimated and used for sampling.

- **SMOTE**

As the previous methods SMOTE starts from a pre-existing dataset and with geometrical operations in the feature space finds new synthetic instances. The idea is to take a point belonging to the minority class, sample one of its *k* nearest neighbours of minority class and find the synthetic point randomly on the joining segment. The effect, comparing the results using a decision tree trained on the original dataset and on the synthetic one, is a better oriented decision region. Comparing with a basic oversampling with replication of the minority class, we have a larger decision region (for the minority class), that leads to a more generalizable model. Indeed, simply replicating the minority class units leads to extra-specific decision boundaries and causes overfitting.

The algorithm takes as input the sample, the number of minority class unit (*T*), amount of synthetic data to be created for each original instance (*N*) and the number of nearest neighbours to be used (*k*). The output is *N x T* synthetic minority class samples.

In each iteration the k nearest neighbour for the current point (*i*) are computed, and one of them is sampled randomly. Then is computed for each attribute (*attr*) the difference (*dif*) between their values and a random number (*gap*) in [0,1] is sampled. The value for that attribute for the new synthetic point is computed as:

$$SyntheticPoints[newindex][attr] = OrigSample[i][attr] + gap * dif$$

SMOTE also shows improvements in correct classification of underrepresented classes. SMOTE– NC (Nominal Continuous) is a variant of SMOTE that consent to use also data with continuous and nominal features, but not full-nominal datasets. We will also add the possibility to introduce nominal features in our datasets.

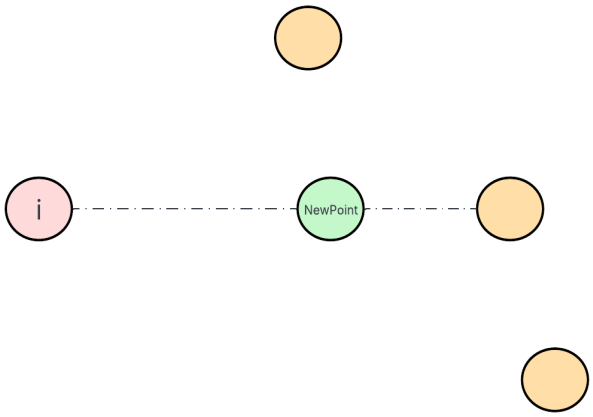

Figure 2 SMOTE process in the feature space

All observed methods are evidently based on manipulating a starting dataset rather than generating synthetic data from scratch. In the next section we will analyse how we overcame this problem to adapt the synthetic data generation to our needs, in this case.

2. Machine Unlearning

We can now explore the methods for Machine Unlearning that we will use in our experiments. A general subdivision of Machine Unlearning methods is between Exact Unlearning and Approximate Unlearning.

Exact Unlearning, even if its aim is entirely removing the data to be forgotten, may be too expensive, and since one of the reasons for the unlearning process is to avoid expensive computations as retraining from scratch, often Approximate Unlearning, that consists in simulating data forgetting, is preferred.

- **Fine Tuning**

Fine Tuning is an Approximate Unlearning technique, that take a pre-trained model and continue its training only on samples from the retain set. An important limitation of this method is that samples from forget test are not truly removed, but training more the model, the influence of forget set on model's parameters is reduced and the loss is minimized further only considering samples from retain set. The aim is to induce the model to tune its parameters only on these samples, removing during new training epochs the old influence of samples that were seen before and now must be forgotten. As said, no information is formally removed from the model, but it is just trained further on samples that can be remembered until (hopefully) samples from forget set have not anymore influence on the model's decisions. The result on some Machine Unlearning evaluation techniques like MIA is that the model after the unlearning process will be much more confident on retain set, since it has seen it longer, than on forget set, that has not seen anymore, and this will be treated almost like an unseen set, that is the objective.

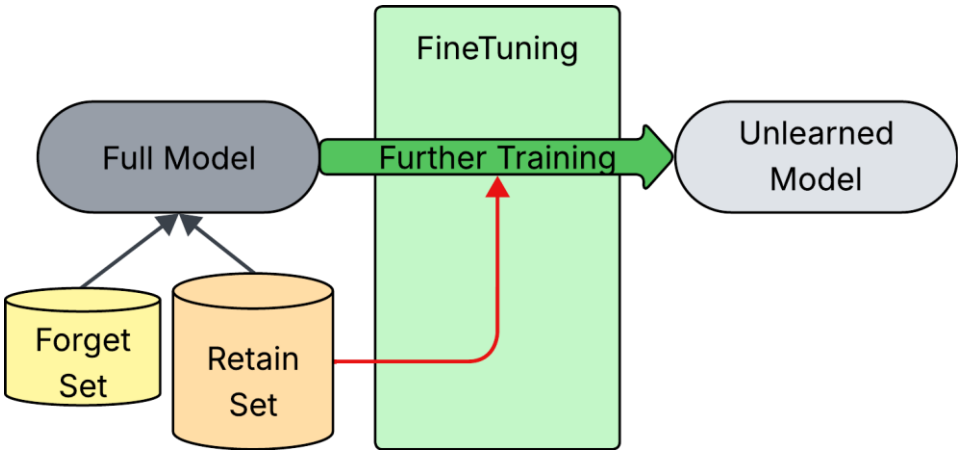

Figure 3 Representation of Fine-Tuning process

- **Negative Gradient Ascent**

Negative Gradient Ascent is a technique based on using Gradient Ascent (same core idea of Gradient Descent, but using maximization instead of minimization as objective) on the loss computed on samples from forget set. The effect on model's parameters is that they are shifted far from the original values learned considering forget set part of training set.

The process is iterative, and it is the opposite of a traditional learning process of a Neural Network, after the evaluation of each batch of forget set samples, the gradient of the loss is computed, and the weights are updated as:

$$\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{\alpha}\, \boldsymbol{g}$$

Where g indicates the gradient of the loss on forget set samples. Notice that the direction of the gradient is followed, instead of its opposite direction as is usual in the standard training process.

The effect is that the performances of the model on the forget set are significantly worsened, and usually a stopping criterion on the accuracy of the predictions on the retain set is utilized, to avoid a worsening also on these samples.

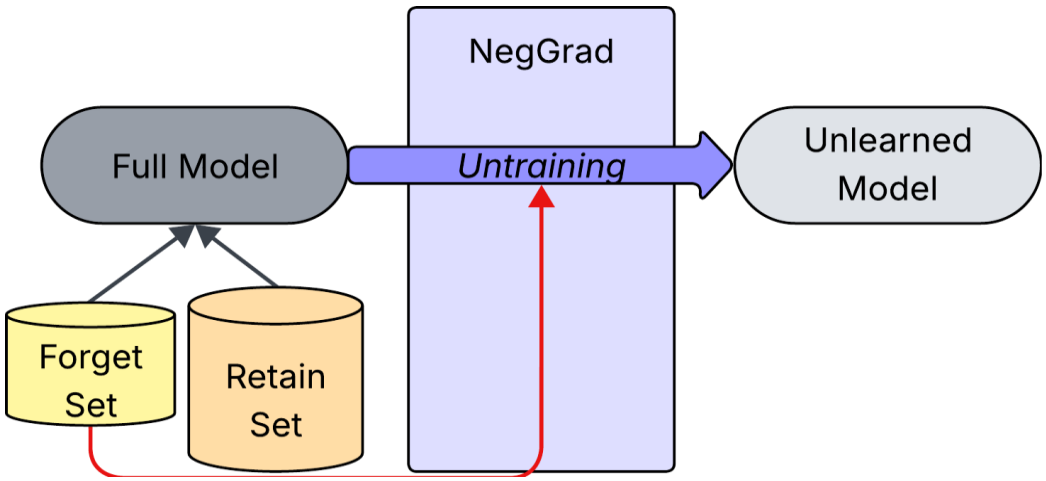

Figure 4 Representation of Negative Gradient Ascent process

- **Advanced Negative Gradient Ascent**

Advanced Negative Gradient Ascent is an improved version of the previous technique. The idea is that not only an *inverted training* (called *untraining* in Figure 3) is performed on forget set like Negative Gradient Ascent, but also a further gradient descent on retain set is applied.

Formally, there is an hyperparameter $\beta \in [0,1]$ that balances the gradient descent on the retain set and the gradient ascent on the forget set, as follows:

$$\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{\alpha}\, [\, \beta\, \boldsymbol{g_r} - (1 - \beta\,) \boldsymbol{g_f}]$$

Where α indicates the learning rate, $\boldsymbol{g_r}$ the gradient computed on the retain set and $\boldsymbol{g_f}$ the gradient computed on the forget set.

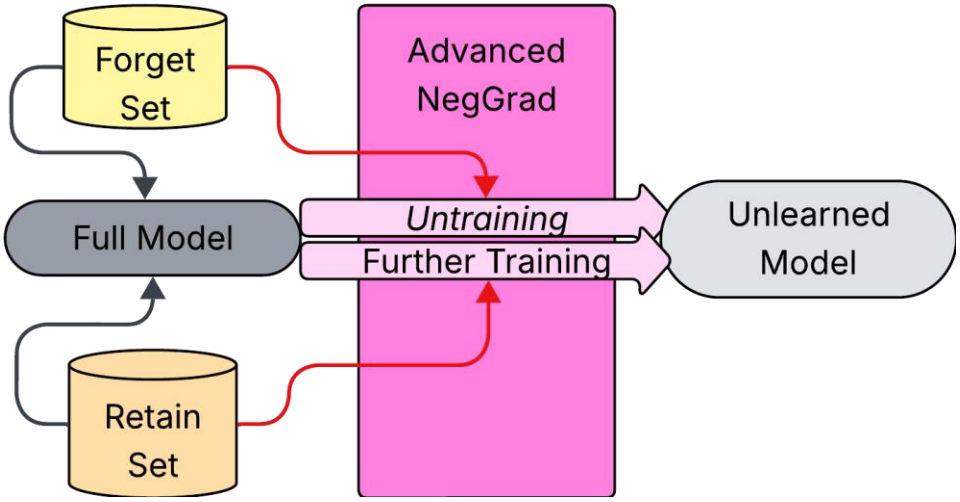

Figure 5 Representation of Advanced Negative Gradient Ascent process

3. Methodology

The purpose of the work is to create Synthetic Tabular Data, and the key point is the possibility of being able to tune some characteristics of the generated data to adjust the difficulty of the machine unlearning process. In general, you will mainly want to control how entangled the data generated for different classes are with each other, and based on this, how difficult it is to classify them. We can control different aspects of the data that is generated, but in general the main concept is how much the distributions overlap or not. Since the way to evaluate the work done is to use a classifier, the result that we generally expect is to have a higher accuracy with less overlapping and more distinct distributions.

Mainly the strategies that had to be chosen regarding the two macro-phases of the process:

• Data Generation: most Data Generation techniques are based on real datasets, we needed to create new data from scratch, so we had to use multivariate Gaussian distributions.

• Classification: there are many methods for classification, in our case not necessarily binary, and the one used is a simple neural network with standard hyperparameters.

Regarding Data Generation, the user specifies the number of columns (attributes), the number of different classes, the number of samples for each class and the desired divergence between distributions, then:

1. An optimizer computes parameters for multivariate distributions of each class minimizing the distance of their divergence matrix with the target matrix specified as input.

2. Instances are sampled from these distributions, generating the synthetic dataset.

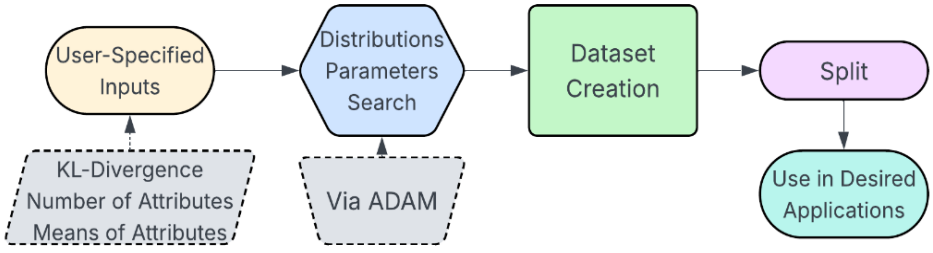

Figure 6 Our Synthetic Data Generation flowchart

1. Classification

For Classification, we used simple Neural Networks composed by different numbers of Fully Connected Layers. We decided to use four different Neural Networks (Small, Medium, Large, Extra Large) to observe results with different complexities. After each Fully Connected Layer (except for the last one) there is a ReLU activation function.

Fully Connected (FC) Layers are one of the most common layer types in Neural Networks. They are composed by neurons, and each neuron of the layer is connected with all the neurons of the previous layer and of the next one.

They are very flexible and have many parameters, thus are often used for classification and regression tasks. Often, they are accompanied by other types of layers and due to their large quantity of parameters are susceptible to overfitting, but in our case the task was simple, since we used not so complex datasets in the shown experiments.

Formally, the output of the layer is $\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{W}$ is the weight matrix, learned during training process, $\boldsymbol{b}$ is the bias (as a vector), f is the activation function. The output is a vector with a value for each of the neurons of the layer. $\boldsymbol{W}$ and $\boldsymbol{b}$ are learned during the training process.

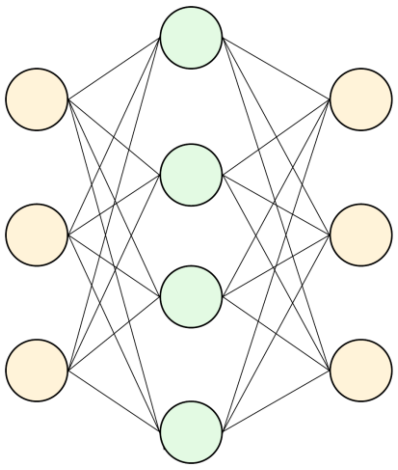

Figure 7 Fully Connected layer architecture

The activation function we used is ReLU, which is defined as follows:

$$ReLU(x) = \max(0, x)$$

Thus, it filters every output smaller than zero and maintains unchanged every output larger than zero. ReLU is not the only type of activation function.

The role of activation functions in Neural Networks is to filter values below some threshold (denoising) and to create non-linearities between layers. Having non-linearities in a Network is useful (and necessary) because it helps to reproduce complex functions.

Without activation functions, a Neural Network, regardless of its depth, would be a simple linear combination of inputs, and many complex relationships would be unlearnable. Sigmoid, SoftMax (for the last classification layer) and softer version of ReLU are other examples of popular activation functions.

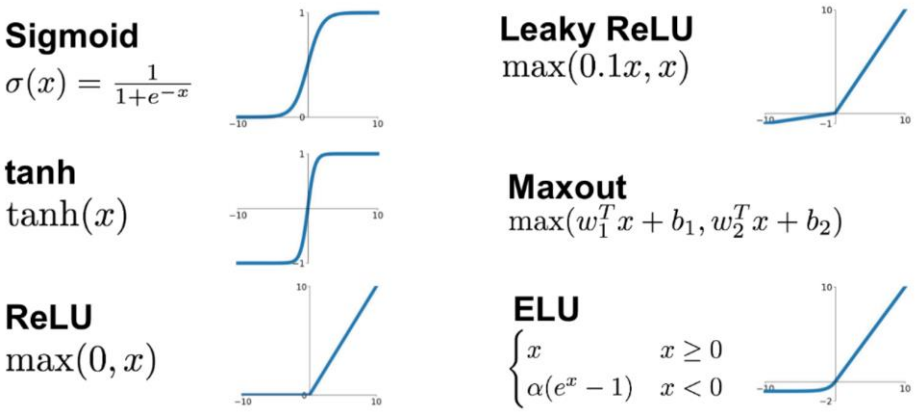

Figure 8 Some popular activation functions

Let us observe deeply the architecture of the Neural Networks we used:

• Small Classifier:

- 6 Fully Connected layers
- ReLU after each Fully Connected layer

• Medium Classifier:

- 8 Fully Connected layers
- ReLU after each Fully Connected layer

• Large Classifier:

- 9 Fully Connected layers
- ReLU after each Fully Connected layer

• Extra Large Classifier:

- 11 Fully Connected layers
- ReLU after each Fully Connected layer

The input size is the number of attributes, while the output size is the

number of classes, as the Neural Network gives a score for each class for each sample that indicate how much it is likely to belong to each one.

The optimizer used for backpropagation and weight updates is ADAM.

ADAM is an optimizer designed to efficiently update network weights during the training process by adapting the learning rate for each parameter individually. ADAM combines the advantages of two other popular optimization techniques: AdaGrad (Adaptive Gradient Algorithm) and RMSprop (Root Mean Square Propagation). Adam calculates adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. Essentially, it keeps track of an exponentially decaying average of past gradients (like momentum) and an exponentially decaying average of past squared gradients (like AdaGrad and RMSprop). Momentum gives inertia to the optimizer, lowering oscillations and leading to a faster convergence.

ADAM is widely used for large quantities of data, particularly in Deep Learning. Formally at each step it is computed the gradient g, then momentum and second moment are updated as:

$$m_t = b_1 m_{t-1} + (1 - b_1) g_t$$

$$s_t = b_2 s_{t-1} + (1 - b_2) g_t^2$$

And weights are updated as:

$$w_{t+1} = w_t - \alpha \frac{\frac{m_t}{1 - b_1}}{\sqrt{\frac{s_t}{1 - b_1}} + \varepsilon}$$

Where α is the learning rate, while b_1 and b_2 are hyperparameters that control exponential decay.

2. Data Generation

Here we have a list of what we can control about our Data Generation:

- Divergence between classes distributions
- Initial Parameters for distributions
- Constraints for Parameters
- Number of Continuous Attributes
- Number of Ordinal Attributes (and number of respective bins)

• Number of Categorical Attributes (and number of respective possible values)

• Number of Samples for each Identity

• Levels of Temperature for Identities

• Number of Identities for each Temperature Level

About Divergence, user can specify a specific target matrix with a different divergence between the distributions of each pair of classes, or a single divergence score for all pairs of classes.

The divergence score is computed with the KL–Divergence via the command:

torch.distributions.kl.kl_divergence(distribution_1 , distribution_2)

The Kullback– Leibler divergence (KL-Divergence) is a non-symmetric score that measures how much are different two distributions. When the two distributions are multivariate it is computed as:

$$D_{KL}(p||q) = \frac{1}{2} log \frac{|\Sigma_q|}{|\Sigma_p|} - \frac{1}{2} \mathrm{E}_p \left[(\boldsymbol{x} - \boldsymbol{\mu_p})^T \Sigma_p^{-1} (\boldsymbol{x} - \boldsymbol{\mu_p}) \right]$$

$$+ \frac{1}{2} \mathrm{E}_p \left[(\boldsymbol{x} - \boldsymbol{\mu_q})^T \Sigma_q^{-1} (\boldsymbol{x} - \boldsymbol{\mu_q}) \right]$$

The KL-Divergence could be interpretated as the expected excess surprise from using Q as a model instead of P when the actual distribution is P.

Then, we pass to the optimizer two arrays:

• Mean Matrix: contains means for each attribute (m columns) for each class (N rows). *[Dimension: N x m]*

• Covariances Array: contains covariances between each pair of attributes for each class. *[Dimension: N x m x m]*

The optimizer will find new values for these arrays that satisfy the desired requirements in terms of constraints and KL-Divergence, then these parameters define the multivariate distributions, one for each class, from which the synthetic data will be sampled.

Since we use Adam optimizer, we can specify an initial point, for example if we want specific values for some distributions, this point is always set to zero to guarantee a better convergence of the algorithm, but it is added in the loss function with respect to KL-Divergence to consider it in computations and also added at the end to have desired parameters. We

could think the initial point for the mean parameters as an indication for the magnitude of the attribute.

We also introduced shallow constraints with a penalty in the loss function of the optimizer which is computed as the MSE between the current solution and the desired one. This penalty is added to the MSE between the current divergence matrix and the target one.

To manage ordinal attributes, the user can specify the number of bins of each attribute, then a column for each ordinal attribute will be generated as a continuous attribute and afterwards they will be discretized in bins. The edges of the bins are computed to have an almost uniform distribution of data among them. Since values for the same attributes come from different distribution (one from each class), it is necessary a function to find quantiles of a mixture of gaussians. The result will not be an exactly uniform distribution among bins because of the dependencies from other attributes.

To manage categorical attributes, user can specify the number of possible categorical values allowed for each attribute, and a column for each possible value for each categorical attribute will be generated as a continuous attribute. Then, for each categorical attribute, for each row the maximum value of the column related to its possible values will be set on 1, others on 0, and each column will become a binary column that reflects if a row belongs to a category or not.

If a preponderance for a categorical value in a class is desired, it is enough to set the mean of the column related to that value higher than other for that class.

The number of samples for each class is not simply the number of rows that are sampled independently from the multivariate gaussians, but it is specified with number of identities and number of samples for each identity.

We consider an identity a set of instances related to the same individual, so to reflect this on our dataset, we distinguish:

• Fixed Attributes: they are the same in all the samples of the same identity. For example, height on adults.

• Minimum Variance Attributes: they are very similar in all the samples of the same identity. For example, weight on adults.

• Standard Attributes: they are independently sampled in different rows related to same identity. For example, current mood of a person.

To implement this, first we sample a single row, then we create other samples for the same identity. From the technical point of view, we acted like this:

• Fixed Attributes: their values are simply copied from the first row in all the samples related to the identity.

• Minimum Variance Attributes: their values are sampled from a multivariate normal centred with the value of the first sample of the identity, no covariances and a variance equals to the mean divided by 100. (To preserve the magnitude of the attribute).

• Standard Attributes: their values are sampled from the original distribution, conditioned to the values already sampled. There is a specific formula to sample form a multivariate gaussian of which some values are already realized.

Regarding temperature, we wanted to admit into the dataset identities that belong to more classes, instead of only one. This effect could be replicated using SoftMax with different temperatures to make different levels of instability in terms of class coherence in an identity. The SoftMax is defined as:

$$softmax(\mathbf{z})_i = \frac{e^{\frac{z_i}{T}}}{\sum e^{\frac{z_j}{T}}} \qquad \text{where T is temperature}$$

The idea is that for each identity a vector of numbers sampled from a uniform distribution between 0 and 1 is generated. Then, this vector is mapped into a distribution over the classes regulated by the temperature T. The higher is T, the closer the output probabilities in the vector will be.

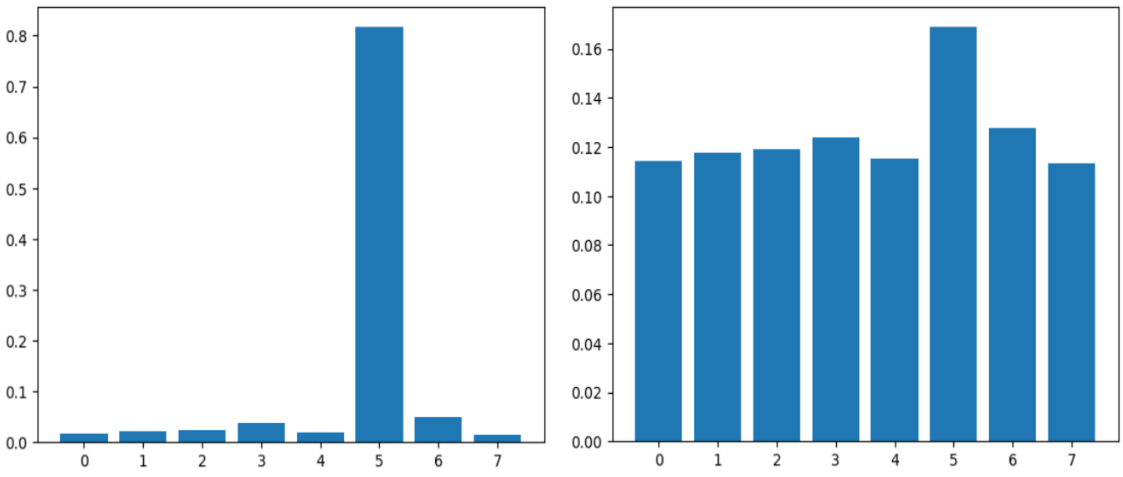

Figure 9 SoftMax without temperature (on the left) and SoftMax with temperature T=10

Thus, through SoftMax a vector of probabilities for classes is created starting from a random vector and a temperature value for each identity,

and every sample belonging to it will have its class sampled from this distribution. Fixed attributes are copied in every sample of the identity from the first row, regardless of their classes (even if they are different from the one of the first row), Minimum Variance attributes are sampled with a reduced variance from the distribution of the class of the first row, but Standard attributes are sampled from the multivariate of the new class (the one of the current sample that generally could be different from the one of the first row of the identity) conditioned to the already sampled values. This create a bit of controlled confusion in the dataset, with samples that are not exclusively from a class, but with attributes that come from different distributions. The higher is the temperature of an identity, the more is likely to change class through its samples. Multiple values of temperature can be set in our algorithm, and user can specify how many identities must be created for each temperature value.

The Synthetic Data experiments were performed with the algorithm described, above. The next section is about how the synthetic data generated were used to perform Machine Unlearning experiments.

3. Data Splitting and Selection

After data generation, for the experiments the dataset is split in some subsets, to perform Machine Unlearning experiments:

• Training set: the set on which the Full Model is trained, it is composed by the 80% of the entire dataset.

- Forget set: the set that must be forgotten by the Unlearned Model, it is composed by the 15% of the training set.
- Retain set: the set that must remain in the Unlearned Model, and on which the Gold Model is trained, it is composed by the 85% of the training set.

• Validation set: validation set for all the models; it is composed by the 10% of the entire dataset.

• Test set: test set for all the models; it is composed by the 10% of the entire dataset.

The way the samples are split among subsets is not completely random, since the idea is not to forget random samples from the original training set, but to remove some specific sample related to some individuals.

Indeed, we want to maintain in the same subset all the samples related to a single identity, because the idea is to remove from the original model the

information about some specific individuals, represented by a specific identity.

Moreover, a specific value of temperature can be indicated for forget set, in that way only identities with that value of temperature will be selected to be forgotten and the user can control how much the forget set is entangled in the entire training set.

The idea is that requiring to forget identities with low temperature, forget set will be less entangled since all the samples related to the same identity will be concentrated closer in the latent space, while high temperatures identities will be more spread and mixed with other identities.

Then, each model is trained on the proper training set, and they are ready for predicting values on unseen samples.

4. MIA

After that, a MIA (Membership Inference Attack) is performed on both models.

We already mentioned what a MIA is, but let us observe it how we used it in our experiments.

Our version of MIA is the black-box one, since the attacker cannot see the model, but only the outputs it gives. Using features as loss, logits or confidence gives more information since not only indicate what is the class that the model is assigning to a label, but also how confident it is. From this information the attacker, that is a binary classifier, tries to understand if each sample was in the training set of the model or not.

In our experiments we used MIA against both Full and Gold model, comparing their predictions on forget set and test set.

While the Gold model has not seen both sets, thus the MIA should not be able to distinguish where samples come from, the Full model has seen the samples from forget set, but not the ones from test set, and we expect a higher accuracy of MIA as result.

Moreover, we tried to provide to MIA linear regressor different combinations of features about predictions on samples. We computed, for the model prediction of each sample:

• Loss

• Maximum Confidence: it is computed taking the maximum values among assigned probabilities for each class, it indicates how much confident the model is in assigning a class to a sample.

• Entropy: it indicates the confusion of the model predicting the class for a sample, the closer are different class probabilities, the higher is the entropy, since it means that the model has not a clear assignment.

• Binary Feature for correct classification or not: it indicates whether the model prediction is correct (1) or not (0)

• Probabilities for each class (Normalized *logits*)

After testing various combinations between these, Entropy and Maximum confidence turned out to be not informative at all, since with or without them the MIA result was exactly the same, while the binary feature for correctness of classification resulted too much informative, leading to a MIA result much higher than 0.5 also for the Gold model, that is not desired.

Which of these features must be used could be subject of discussion and the user can choice how to combine them depending on how much power is desired to give to the MIA, the more features it has, the more it will be capable to capture differences between samples predictions. Since in our examples datasets are not very complex or large and models have only few layers, we decided in our experiments to utilize only the loss, that gave us satisfying results.

Another test was performed using the outputs of the Neural Network before the last layer (the classification one), in order to capture the position of the sample in the latent space, but also this resulted to be too much informative, and providing this information the MIA regressor was able to split too much efficiently test and forget set also for Gold model. Probably this behaviour is due to the selection of the identities to be forgotten, that are picked just among the identities of a single temperature level. Using in a forget set only identities from a subset of the dataset could create patterns that MIA could exploit to divide members and non-members of the training set, even if the Gold model has not seen both.

The output of MIA is the accuracy, that indicates the proportion of samples of which the provenance was correctly identified, and forgetting score, that is the distance of the accuracy from the random guess (0.5). This is because if the accuracy of MIA is lower than 0.5, it means that it could be used as an inverted classifier, and consider positive predictions as negative and vice versa.

Thus, the higher MIA accuracy is for the Full model, the more evident is if a sample was or not in its training set, while for the Gold model we always expect a result near 0.5, because it truly has never seen both samples, and we are asking to the binary classifier of MIA an impossible challenge.

We also notice during testing that observing accuracy of models could be informative about interpretating MIA results. Indeed, if the model learned very well and have both good test accuracy on unseen samples, even without the forget set the Gold model can still perform good and be able to deceive MIA, having a good capability to predict unseen samples as the Full model. However, on the contrary, if models have not learned very well the training set, their prediction will be near random guess, and it would be impossible for the MIA classifier to distinguish unseen and forget data also for the Full model, because its prediction would be too poor and similar to a random assignment, even on seen samples.

In both these cases the difference between MIA results on Full and Gold model is not significative, and the Unlearning process has not much sense. Indeed, we are interested in the Delta MIA Accuracy between the two models, or formally equivalently the Forgetting Score of the Full Model. (Notice that in our experiments we will show Delta Forgetting Score in our plots, since in practice the true value of MIA accuracy on the Gold Model is slightly different from the theoretical value of 0.5). This value indicates what we can call the unlearning potential of the model, the more this delta is wide, the more the unlearning process can work. Indeed, if the MIA accuracy of the Full model is high, it means that it is easy to understand if a sample was in its training set or not, and the unlearning process has room to act. However, a lower difference of MIA accuracy between Gold and Full models also suggest that the Full model already provides a good level of privacy of its training set, on one hand this means that the unlearning process is less necessary, on the other hand that the Unlearned model will be more efficient than the Unlearned model of another model that guarantees less privacy, because it would start from a already good model in terms of safety.

The MIA accuracy for the Unlearned model will be included between the two MIA accuracies of Full and Gold model. The same holds for the Forgetting Score.

The same concepts will be analysed and discussed more deeply in the Experiment section.

5. Unlearning Evaluation

The last part of our work was to apply the presented unlearning techniques to our dataset, and using the MIA accuracy to evaluate results.

The experiments and the results found will be discussed more precisely and deeply in the proper section, while here we will just discuss briefly the process and the reasons of our choices.

First, as showed in the previous section, we tested model with different complexity with MIA to observe if, as expected, a decreasing trend is followed by the leakiness of the trained classifier with respect to an increasing KL-Divergence.

Then, we applied different unlearning techniques to verify that the unlearned model guarantees a better privacy with respect to the original Full model, that means having a Forgetting Score included between the Full model one and the Gold model one, that is theoretically zero (not always in practice since the linear regressor of MIA could capture some hidden patterns in the forget set, especially if, as in our case, it is not selected randomly from the training set but from a subset of it).

The result, as will be discussed in the next sections, confirmed what we expected, and synthetic data generated has been proven successfully to be suitable for experiments in Machine Unlearning, with the advantage of being able to be controlled accordingly to necessities and providing a perfect level of privacy, and as benchmarks to test and compare different unlearning methods.

4. Data Synthesis Experiments

Before Machine Unlearning experiments, we tested our Data Generation algorithm with different parameters, to check its correctness and proper functioning. We also tested it to observe elapsed time and how does it scale changing features as number of classes, number and type of attributes and number of samples and identities.

For the runs we used T4 GPU from Google Colab, and for the initial Data Generation experiments a simple classifier of three layers to observe the accuracy of classification performed on our synthetic dataset. As will be explained later, for more complex experiments new models with different complexity will be used.

In the first part of the tool development process, we performed experiments in different phases observing at each step results before adding new features and obtaining the complete version. Here these experiments are shown, before analysing the final experiments that conclude our work.

Even if the key aspect of the work is about Data Generation for Machine Unlearning, testing our algorithm for pure Data Generation was useful to become aware of which hyperparameters of our dataset are important to increase or decrease its complexity, and how much each feature impacts on the difficulty of the task for the classifier.

Generally, we could expect that with higher KL-Divergence the accuracy will grow, because having more separate classes distributions will make it easier for the classifier to get predictions right.

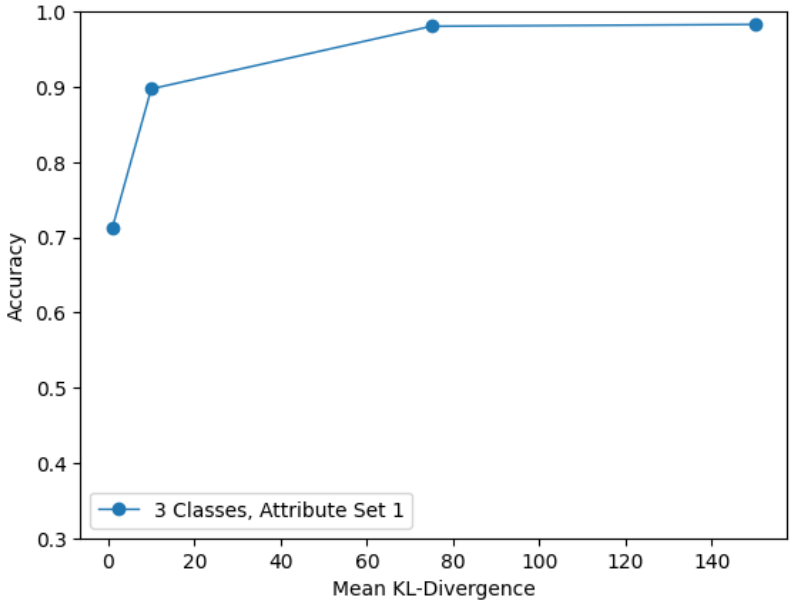

Figure 10 Example 1 of model accuracy on Synthetic Dataset

The example in **Figure 10** was performed with:

- 3 Classes
- 3 Continuous Attributes (Initial Point: [170, 65, None], No Constraints)
    - First is Fixed, second is Minimum Variance
- 1 Ordinal Attribute (5 bins, No Initial Point, No Constraints)
- 1 Categorical Attribute (3 values, No Initial Point, No Constraints)
- 300 Identities per Class, 10 Samples per Identity

We can observe some results for the 150 KL-Divergence synthetic dataset:

- Matrix of Means:

0.58	-0.04	0.74	1.65	1.7e+02	6.5e+01	1.74
0.85	-0.56	0.58	-0.61	1.7e+02	6.5e+01	-2.07
-1.83	-0.56	-1.08	0.26	1.7e+02	6.5e+01	1.6

Table 1 Result of Means Matrix for dataset with 150 KL-Divergence between classes

- Divergence Matrix:

0.	150.73770142	150.53108215
150.10214233	0.	149.41575623
151.18327332	151.38252258	0.

Table 2 Result of Divergence Matrix for dataset with 150 KL-Divergence between classes

As we can observe, desired divergence is reached, and the attributes reflect the chosen initial point (in the matrix of means we have ordinal, categorical

and continuous attributes, in this order, so we can see that 170 and 65 are respected for all classes).

As expected, when KL-Divergence is higher, the test accuracy increases.

For the search of optimal parameters of the distributions different tolerances can be used, depending on specific needs or values scale.

In the next example, we kept all parameters equal, except for the number of classes, that we raised to 5:

- 5 Classes
- 3 Continuous Attributes (Initial Point: [170, 65, None], No Constraints)
    - First is Fixed, second is Minimum Variance
- 1 Ordinal Attribute (5 bins, No Initial Point, No Constraints)
- 1 Categorical Attribute (3 values, No Initial Point, No Constraints)
- 300 Identities per Class, 10 Samples per Identity

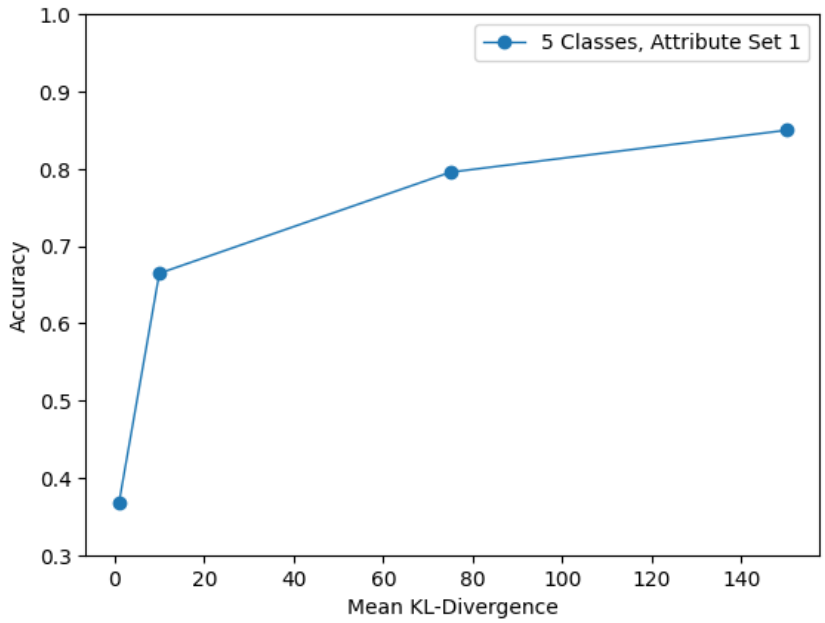

Figure 11 Example 2 of model accuracy on Synthetic Dataset

As we might have expected, with more classes general accuracy is lower, but the general increasing trend of accuracy with respect to a larger KL-Divergence holds.

The next example shows how, keeping the same number of classes, using more attributes the general accuracy of the classifier increases again, because we are giving more information to the model about samples. It leads to a simpler dataset.

- 5 Classes
- 5 Continuous Attributes (Initial Point: [170,65, None, None, None], No Constraints)
    - First is Fixed, second is Minimum Variance
- 5 Ordinal Attribute (7 bins, 5 bins, No Initial Point, No Constraints)
- 5 Categorical Attribute (5 values, 3 values, No Initial Point, No Constraints)
- 600 Identities per Class, 5 Samples per Identity

Let us denote that just 5 samples per identity are provided, so while the attributes are many, the information about a single identity is not very much. Despite of this, the accuracy results still quite high.

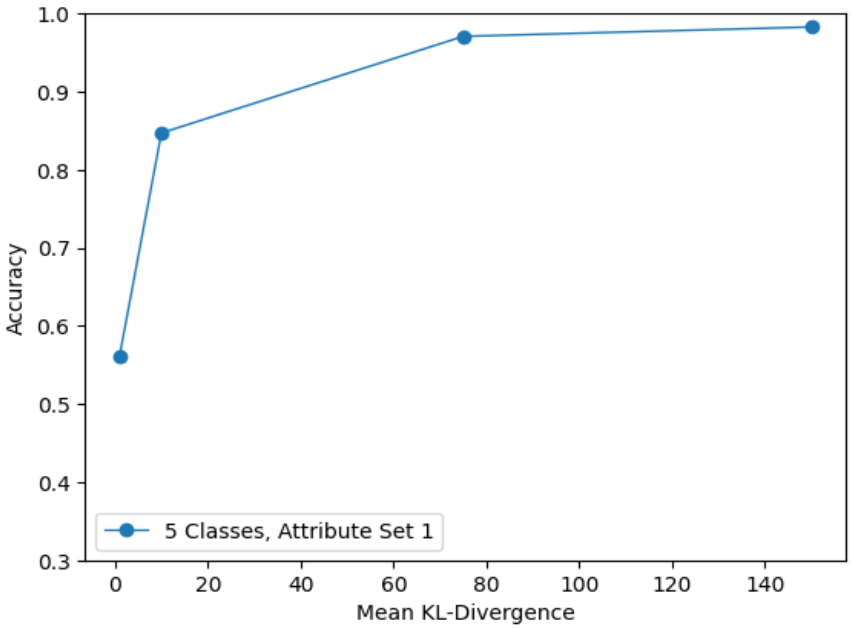

Figure 12 Example 3 of model accuracy on Synthetic Dataset

From now on, all the shown experiments are performed with the final version proposed of the tool.

We can also provide a portion of a generated dataset, in order to effectively see how synthetic data appears. We will use these hyperparameters:

• Number of Classes: 3

• Number of Identities for each temperature level: [5,10,5]

• Temperature Levels: [0.01,1,5]

• Temperature Level for Forget Set: 0.1

• Number of Samples per Identity: 200

• Number of Continuous Attributes: 6 (No Initial Point, No Constraints)

• Number of Ordinal Attributes: 0

• Number of Categorical Attributes: 0

• Number of Fixed Attributes: 2

• Number of Minimum Variance Attributes: 2

• Number of Standard Attributes: 2

• Variance Ratio: 100

We can observe some samples of different KL-Divergence values:

0.455974	0.379259	-0.5553	0.7372	-0.90434	-1.4758	1
0.455974	0.379259	-0.5119	0.5858	0.28408	-0.3831	0
0.455974	0.379259	-0.6179	0.8020	-0.22037	-1.0677	0
0.455974	0.379259	-0.5686	0.8210	0.025825	-1.1221	1
0.455974	0.379259	-0.5607	0.6601	-0.15184	-0.1684	1
0.455974	0.379259	-0.4439	0.6818	0.98455	-0.7948	1

Table 3 Example of Dataset with KL-Divergence 3.0

We can notice from the first six rows of the dataset that, as expected, fixed attributes (first two columns) are kept constant as this is a single identity, minimum variance attributes (third and fourth columns) are similar but not identical and the standard attributes are completely different in each sample. It could also be interesting to observe labels of these samples, that are provided in the last column. Since this is a low temperature identity (level 0.01) we could deduct from this row that the main class of this identity is 1, but still some samples belong to other classes, as desired.

Now, since identities are sorted, we can search for a high temperature identity to observe the difference:

1.06019	1.0274	-0.604759	1.20467	-0.08242	-2.21113	1
1.06019	1.0274	-0.694080	1.44677	-0.48284	-1.14367	2
1.06019	1.0274	-0.70930	1.25305	-0.75031	-1.03831	0
1.06019	1.0274	-0.642710	1.18547	-0.30387	-0.94950	1
1.06019	1.0274	-0.635953	1.23215	-0.18861	0.46567	0
1.06019	1.0274	-0.681238	1.20447	0.37751	-0.96584	2

Table 4 Example of high temperature identity for dataset with KL-Divergence 3.0

Despite the main class of this identity is 1, as we can deduce since the first sample of an identity identifies its main class, across other samples labels are almost uniformly distributed, that is what we expect from a high temperature identity.

Here we can observe instead some plots of synthetic data generated to notice the KL-Divergence effect:

The example in Figure 14 was performed with 3 classes, 3 attributes without initial point or constraints and without identities, the KL-

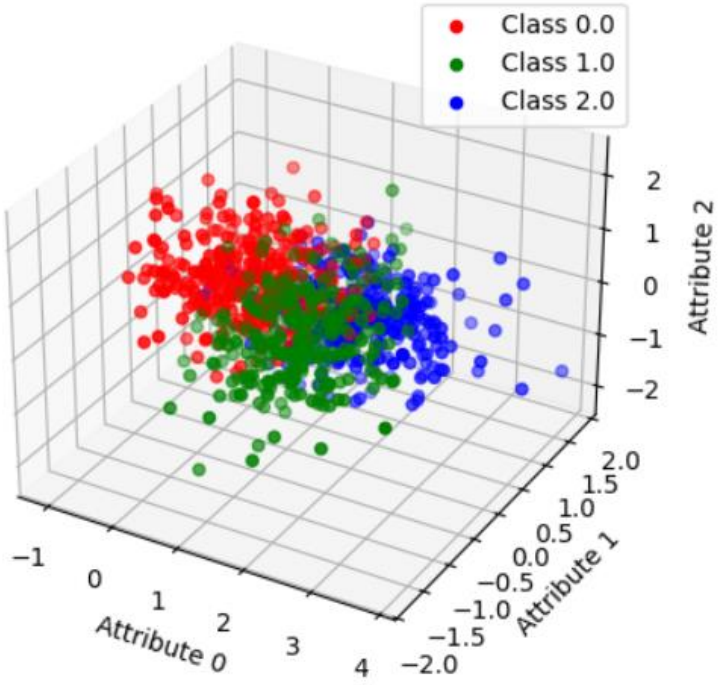

Figure 13 Example of Synthetic Dataset with KL-Divergence 3

Divergence chosen was 3. We can clearly distinguish the three classes.

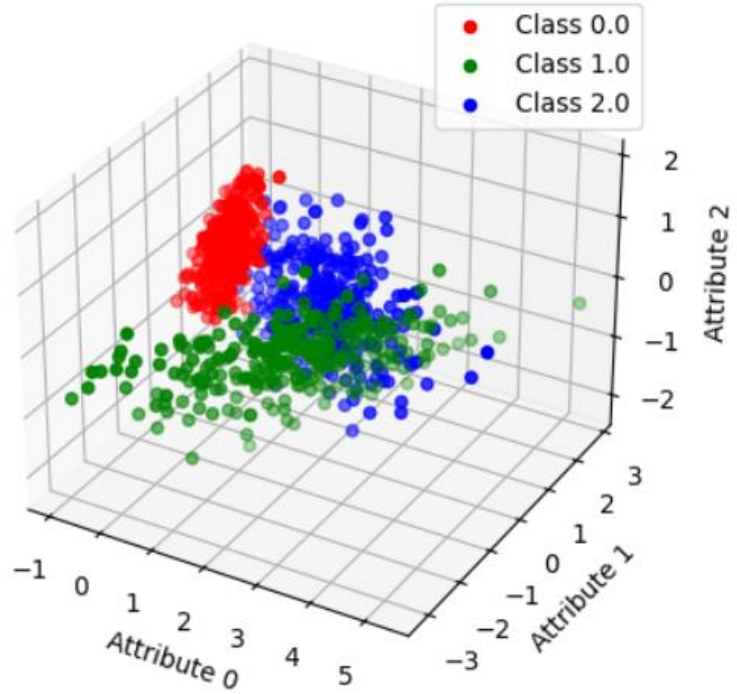

Figure 14 Example of Synthetic Dataset with KL-Divergence 50

Now from Figure 13 we can observe a plot of a synthetic dataset built with the same number of classes and attributes but a higher KL-Divergence. The KL-Divergence chosen is 50.

Here we can notice that classes are more distinguishable with respect to the previous case, and this is exactly the effect that we expect from choosing a larger KL-Divergence.

Now we can observe an example with the concept of identities, to observe how the data distribution changes.

The next experiment was performed with 3 classes, 3 attributes, KL-Divergence 50 and only 10 identities with 100 samples each and maximum temperature level (that means 5) to clearly observe them. To give sense to the concept of identities we fixed the first attribute and set the second one on minimum variance.

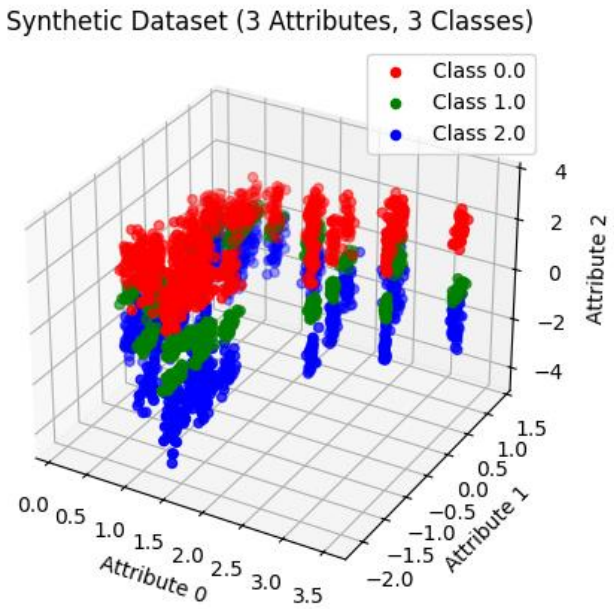

Figure 15 Example of Synthetic Dataset using Identities concept

Since the fixed attribute is the first (*Attribute 0*) and the one with minimum variance is the second (*Attribute 1*), it is clear that each identity is composed by a "column" of points along the axis of *Attribute 2*. Setting their temperature to 5 (the highest level), what we expected was an almost uniform distribution of the samples of the identity among the three classes, and how we can observe from the plot this is confirmed.

Now let us observe a similar plot with the same hyperparameters except for the identities' temperature, that will be set on the minimum value, 0.01.

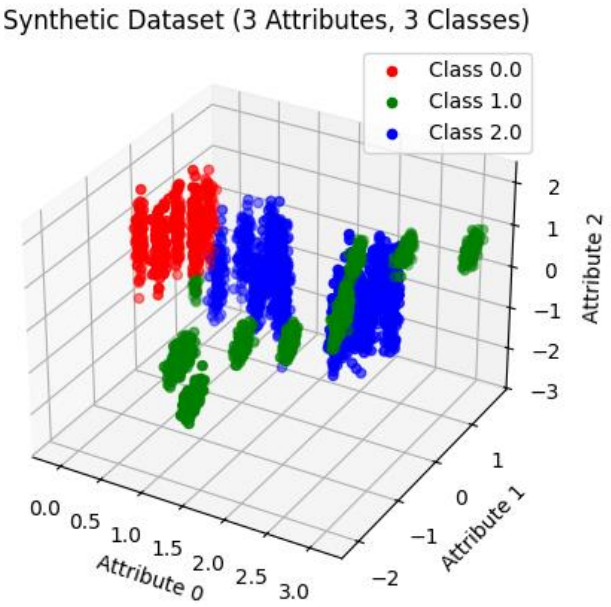

Figure 16 Example of Synthetic Dataset with identities with low temperature

Comparing this plot with the previous one it is evident how now the identities are still evident as "columns" of points, but in each identity, there is almost only one class, that is exactly what we expected to see lowering temperature level.

There is total freedom of choice for the parameters of the Synthetic Dataset, the limitations are only about computational time and secondarily memory.

Obviously, the time taken by the generation of the dataset depends on the desired dimension. Experiments shown that number of samples does not take much time, thus it is not about the size of the dataset, but more concerning number of classes and attributes. This is because the most time-expensive part of the generation is the optimization of distributions parameters, that consist in a matrix of the means of the distributions (with dimension $N \times m$) and a three-dimensional array of the covariances matrices between attributes of each class (with dimension $N \times m \times m$), and results in the order of Nm^2.

Indeed, adding attributes increase more the required time than adding classes.

However, the required time for Synthetic Data Generation is satisfying, and to give an idea datasets of the dimensions we used in our examples composed by 3 classes and 6 attributes took an amount of 20-30 seconds

for the parameter optimization and 10-15 seconds for sampling (using 150 identities with 200 samples each), and must be considered that also the desired KL-Divergence could impact this, since the target of the optimization task can easily influence the taken time.

5. Unlearning Experiments

For the unlearning experiments section, we implemented the three unlearning techniques mentioned in the Methodology section.

The aim of the experiments is to show performances of different unlearning methods on our synthetic datasets and confirm that evaluating the process with metrics like MIA accuracy, the result will be included between the Gold model performance (perfect privacy) and the Full model one (original level of privacy).

The aim for the Unlearned model is to have a MIA accuracy the closer to 0.5, that means that the unlearning process was perfectly successful and deduce whether a sample was in the training set of the model or not is nearly impossible. Here we can observe an example of MIA results:

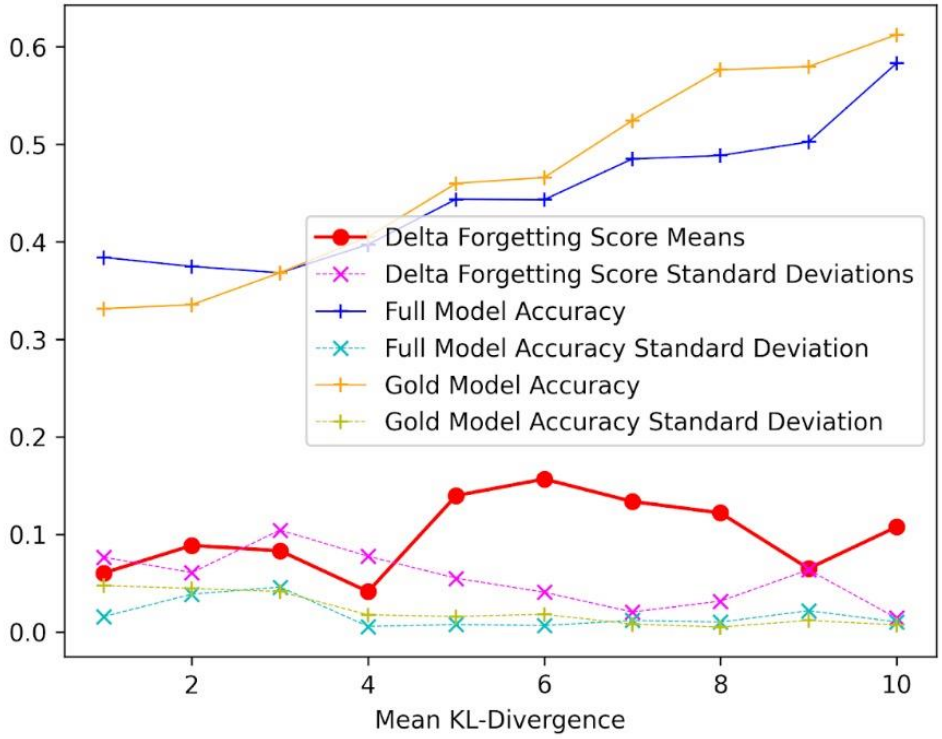

Figure 17 MIA and accuracy result analysis

How we can observe, results are provided as means and standard deviations of Forgetting Scores, because the same experiment is repeated with different seeds in order to reduce the noise due to dataset splits and model weights initialization. Indeed, different splits could lead to different performances, even if the forgotten identities have the same temperature level.

The number of seeds used is another hyperparameter decided by the user, the more seeds are used, the less influent will be the noise impact.

Another decision that the user can make is to keep the same fixed dataset for all the run with different seeds, or not. Even the dataset generation can be different with the same hyperparameters because of the random initialization, and keeping it fixed help to reduce noise.

The hyperparameters used for this example are:

• Number of Classes: 3

• Number of Identities for each temperature level: [10,20,10]

• Temperature Levels: [0.01,1,5]

• Temperature Level for Forget Set: Low

• Number of Samples per Identity: 200

• Number of Continuous Attributes: 6

• Number of Ordinal Attributes: 0

• Number of Categorical Attributes: 0

• Number of Fixed Attributes: 2

• Number of Minimum Variance Attributes: 2

• Number of Standard Attributes: 2

• Variance Ratio: 100

• Model Used: Large_Classifier

• KL-Divergence values: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

• Number of Seeds: 6

• Fixed Dataset: True

From Figure 17, where we used low values for KL-Divergence, and in a small interval, we can notice that the difference between Gold and Full model in terms of Forgetting Score, considering some noise, is almost constant with respect to KL-Divergence.

We can notice, observing the log of this experiment, that generally the accuracies are higher with KL-Divergence equal to 9, because the dataset is simpler, and we can focus on Full MIA accuracy, that is larger than Gold MIA one (that is exactly 0.5, as expected).

This could be due to overfitting, as the higher Gold accuracy (performed on test set) in the 9 KL-Divergence case suggests. If the Full model overfits

data, it is much more confident in predicting them with respect to the unseen ones from the test set, and the MIA regressor can easily distinguish them.

However, this experiment shows that using small values for KL-Divergence is not very informative, since it is not recognizable a clear pattern or trend, and we tried to explore further using larger values.

Thus, we can observe what happens using KL-Divergences in a larger interval, for example from 1 to 100:

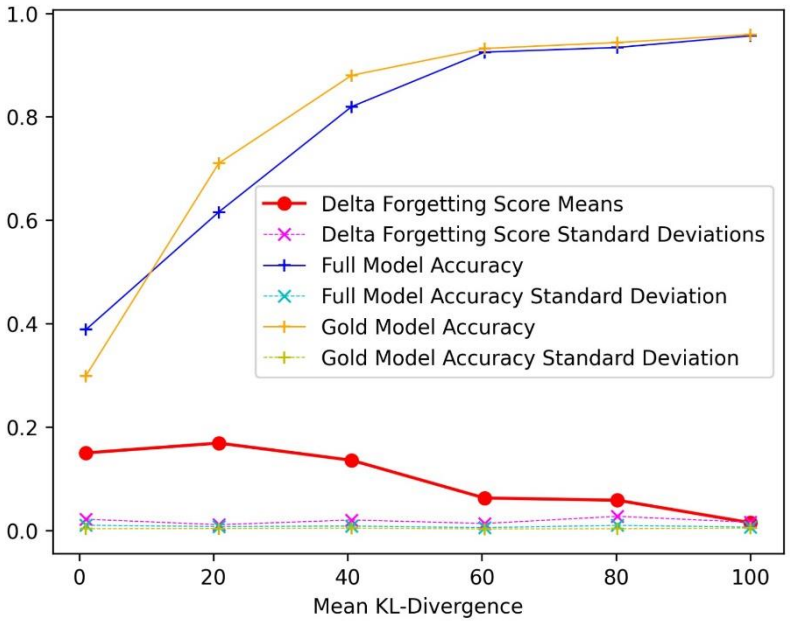

Figure 18 Results of MIA with KL-Divergences from 1 to 100

Observing Figure 18, we can notice that increasing KL-Divergence the difference between Forgetting Score of Full model and Gold model decrease. These results suggest that when the dataset become simpler, models improve their generalization capability and is not necessary to stick to the precise training set characteristics to learn. Thus, models generalize well also on unseen samples and MIA cannot distinguish anymore seen and unseen samples.

From this plot we can understand the aim of the Unlearning process that we will do: with higher KL-Divergences the Full model already has a good level of privacy, if we apply an unlearning algorithm probably we will have a final Forgetting Score lower than applying it to a model trained on a lower KL-Divergence dataset, but it would not have much room to act. With lower KL-Divergences instead the final Forgetting Score after unlearning algorithm will probably be higher (thus worse), because the interval in

which it could stand is larger, but, since the original Full model has not a good level of privacy, it has more sense to be applied

First, we will observe and discuss how models with different complexity perform on the same fixed dataset, to compare them and deduce where and how the unlearning process could have sense to be applied.

The fixed dataset that will be used for all the unlearning experiments was created with these parameters:

• Number of Classes: 3

• Number of Identities for each temperature level: [10,20,10]

• Temperature Levels: [0.01,1,5]

• Temperature Level for Forget Set: 0.1

• Number of Samples per Identity: 200

• Number of Continuous Attributes: 6 (No Initial Point, No Constraints)

• Number of Ordinal Attributes: 0

• Number of Categorical Attributes: 0

• Number of Fixed Attributes: 2

• Number of Minimum Variance Attributes: 2

• Number of Standard Attributes: 2

• Variance Ratio: 100

• KL-Divergences: [1, 20.8, 40.6, 60.4, 80.2, 100]

• Number of Seeds: 3

• Fixed Datasets: True

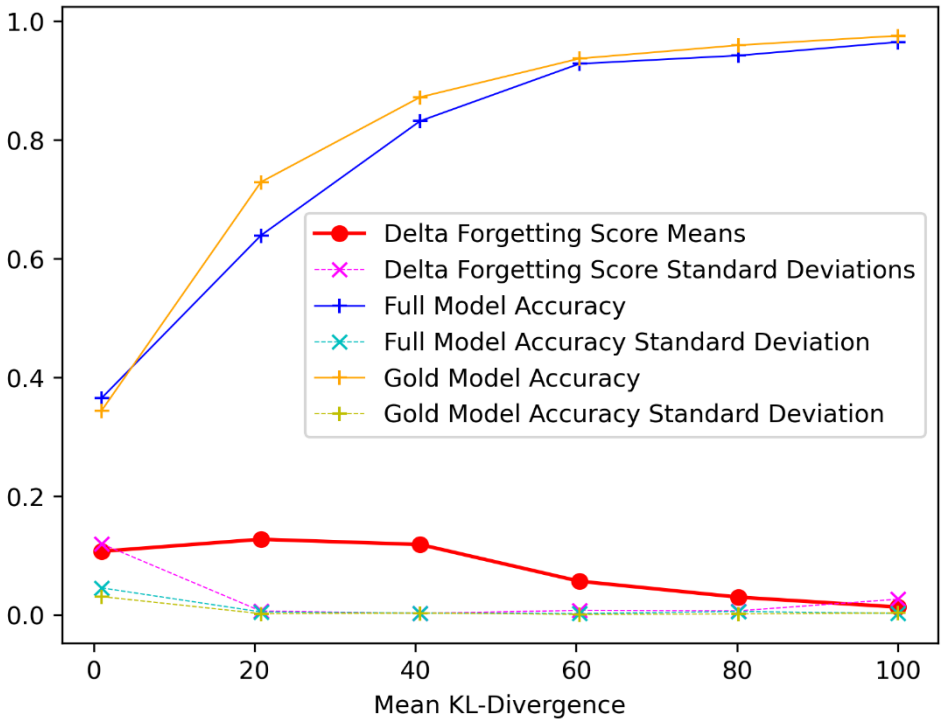

Figure 19 MIA results with Small Classifier forgetting identities with Temperature = 0.01

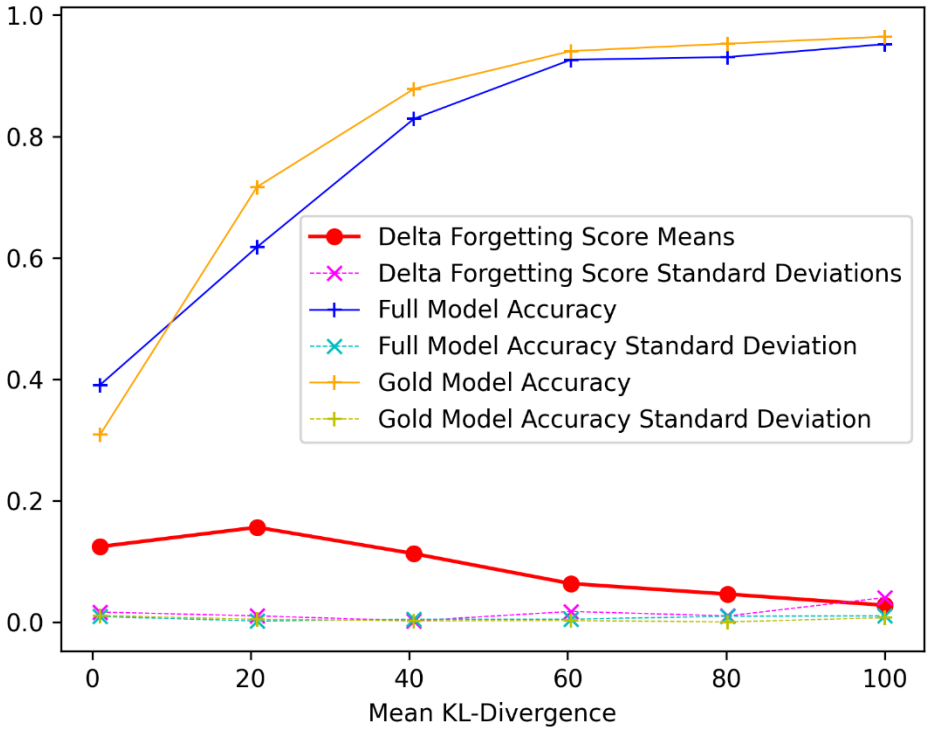

Figure 20 MIA results with Medium Classifier forgetting identities with Temperature = 0.01

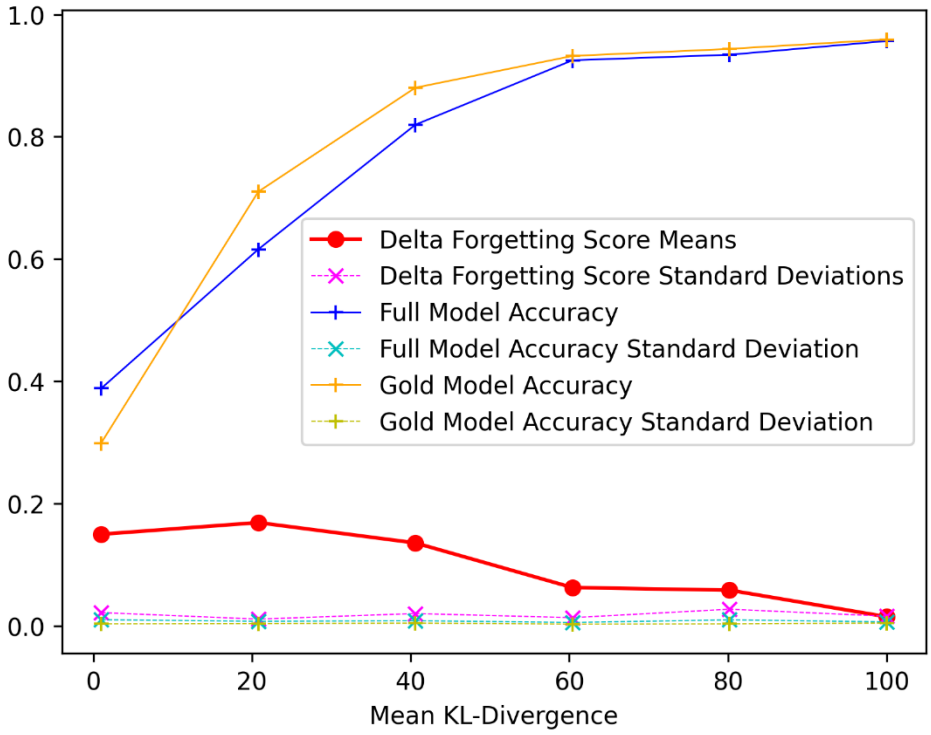

Figure 21 MIA results with Large Classifier forgetting identities with Temperature = 0.01

Observing these plots, we can notice that the deeper is the model, the less is the guaranteed privacy, as is shown by the increasing Delta Forgetting Score. This is an expected result because more layers means that the model learns better its training set and MIA can distinguish with more success whether the classifier has already seen a sample or not.

We can also observe that, as said before, generally using all classifiers, with higher KL-Divergences the Full model already has less dependence from the training samples, that means that a better level of privacy is guaranteed. Indeed, thanks to the increasing simpleness of the dataset, the generalization capability improves and model's performances on unseen samples become as good as predictions on already observed samples.

Before observing unlearning processes results, we can analyse what happens if we change the temperature level of the identities that must be forgotten. All the next plots were made using the Large Classifier.

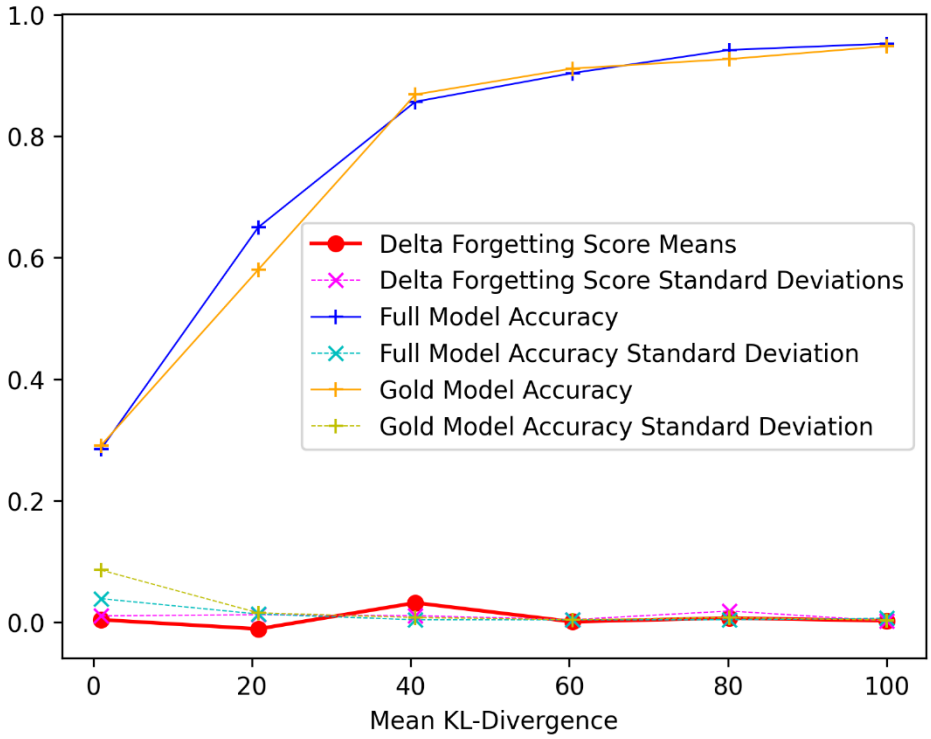

Figure 22 MIA results with Large Classifier forgetting identities with Temperature = 1

From this plot it is evident that using identities with a higher level of temperature in the forget set, the MIA is much less accurate, and the difference between forgetting scores of Gold and Full models is much lower than the previous case.

This was an expected result, as with respect to comparing performances on a simpler forget set composed by more predictable identities with predictions on unseen samples, predictions on identities that require more generalization capabilities of the model are necessarily less confident, and MIA can't distinguish as well as before seen and unseen samples.

In the next plot we will observe what happens when the forgotten identities are even less predictable, using the higher level of temperature in the dataset.

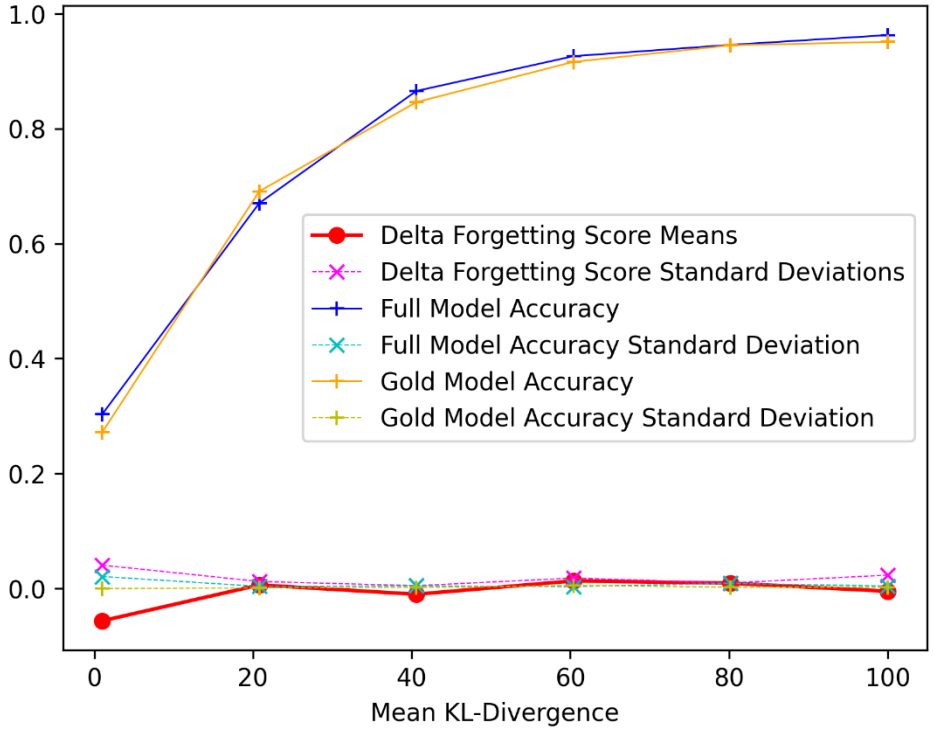

Figure 23 MIA result for Large Classifier forgetting identities with Temperature = 5

As expected, increasing even more the temperature of selected identities for the forget set, the difference between forgetting scores almost disappears, and we can observe almost just noise. We can also notice, observing the accuracies of the models, that they almost coincide each other. This confirms that training two models with training sets that differs of very noisy identities leads to almost identical results. That is explained by the fact that very noisy identities are less meaningful, and the model can't learn very much from them, since four attributes out of six are constrained by the main class of the identity and only two *free* attributes are not enough informative to explain the class label.

Since it has less sense to apply unlearning on the last two cases, results on experiments with lower temperature identities in the forget set will be shown.

Now we will observe the results of the application of the aforementioned unlearning processes on our full models. The experiment will be repeated for three levels of complexity of the models: Small, Medium and Large.

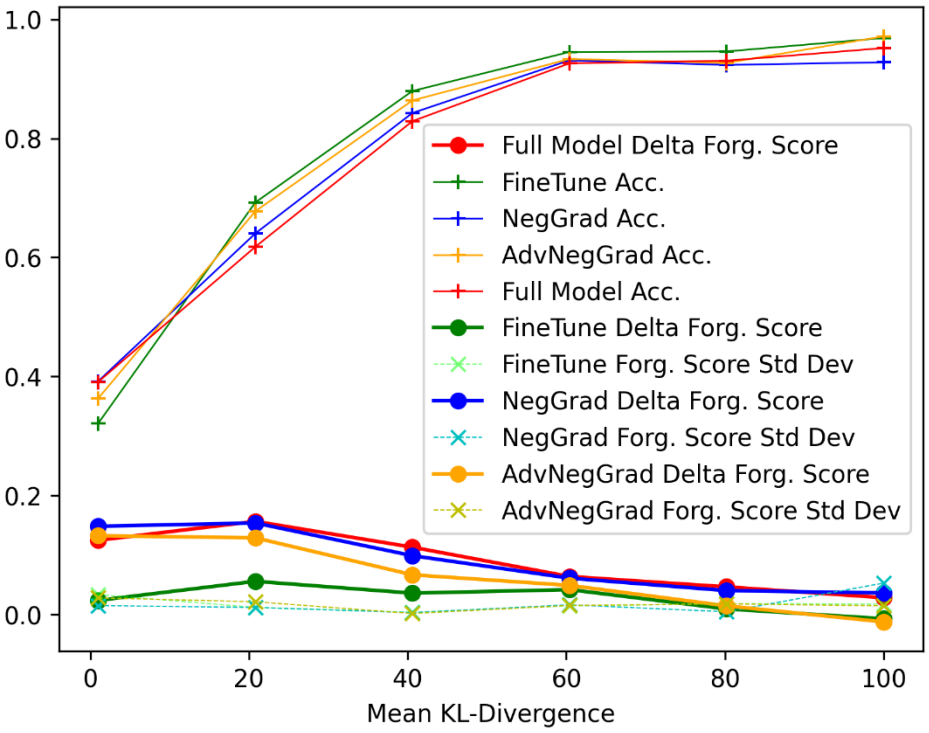

Figure 24 Unlearning results with Medium Classifier

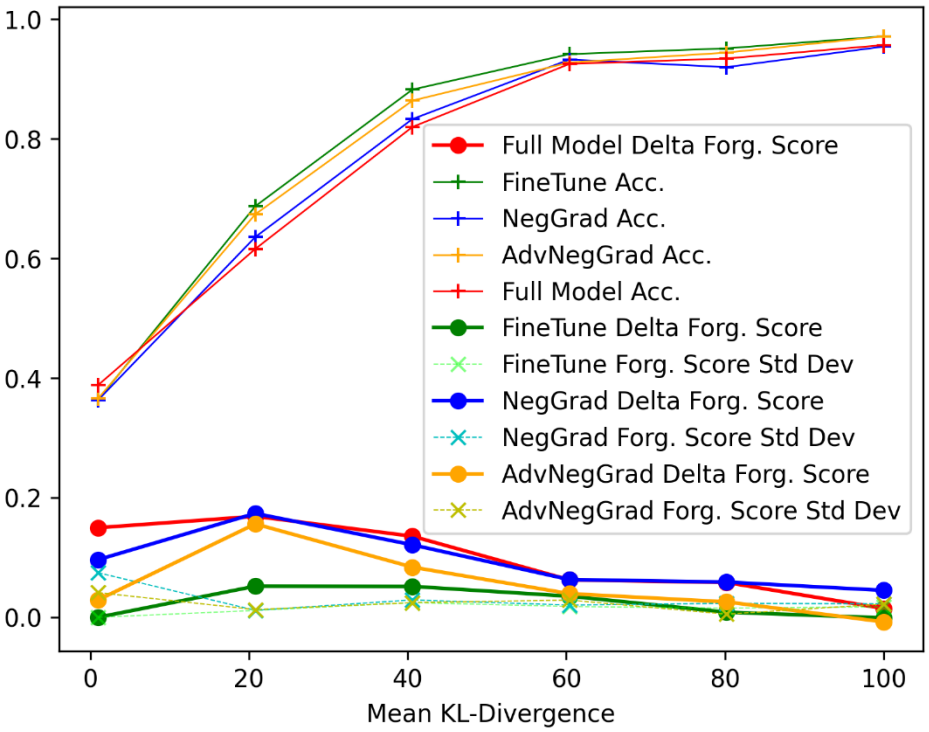

Figure 25 Unlearning results with Large Classifier

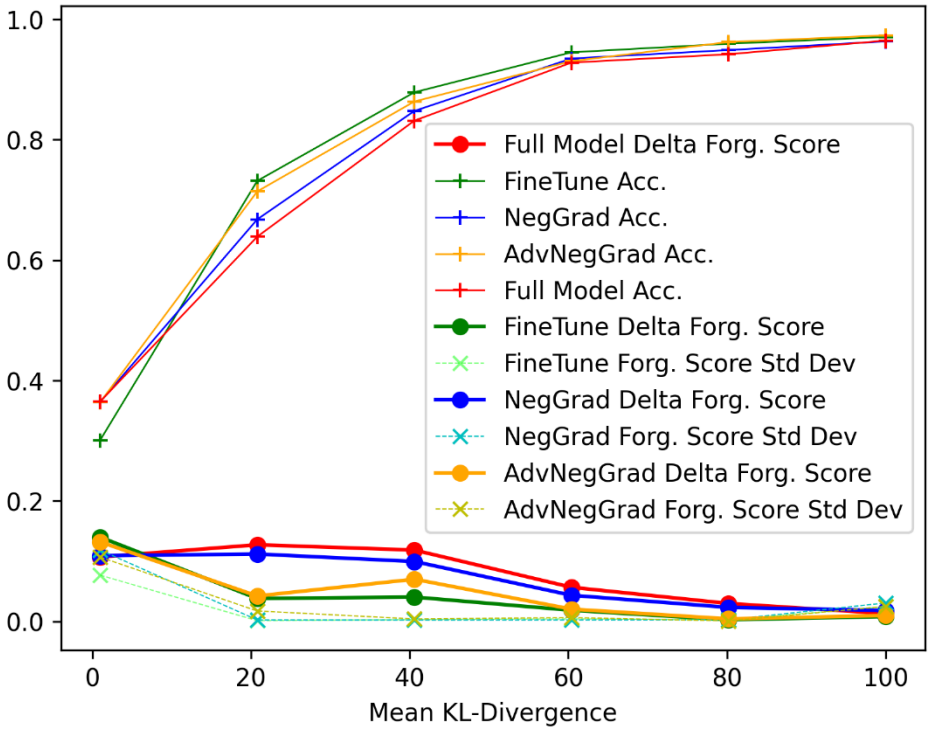

Figure 26 Unlearning results with Small Classifier

As we expected, after the unlearning process the distance of the Forgetting Score with respect to the Gold model's one is reduced, considering some noise especially when the KL-Divergence is very little and the neural network too simple, because in this case classes are almost overlapped, and the entire dataset is close to be pure noise for a classifier that has not enough representative power.

Accuracies are very similar because even if the complexity of the models are different, there are just few layers of difference between them, and the test set (on which the accuracy shown is computed), but it is still meaningful to observe that because we don't want only a smaller Delta Forgetting Score, but also that the Unlearned Model has still a good performance, or even a random classifier would be satisfying.

From these plots it is evident that a comparison between different unlearning processes is easy to make with synthetic datasets, and there is total freedom of choice for the desired models, even deeper or more powerful than the ones used for the examples.

6. Conclusions

We have discussed how we managed to create a tool for Synthetic Data Generation with the possibility to control almost every characteristic, and reasons for its application and uses.

We also shown its application in controlled Machine Unlearning applying different techniques and observing the suitability of our tool for creating easily, quickly and efficiently benchmarks of various types in term of size, entanglement and variety for testing models and unlearning strategies.

We noticed that main limitations of this method are very small values of KL-Divergence, where is difficult to understand how good the result is as classifiers struggle to have good performances, computational time and memory resources if very large datasets in terms of number of classes or attributes are required, since the parameters to be optimized may be too many.

We also tried to stress our algorithm in order to observe its behaviour, for example with very asymmetric values for KL-Divergence among distributions, or using constraints that contradicted desires KL-Divergence values. In the first case the result was still good, without any differences with respect to the symmetric KL-Divergence case, while in the second case, due to the impossibility of satisfying both the requests (it would be mathematically impossible control with total freedom the divergence of distributions with locked parameters), the optimizer simply prefer to satisfy the divergence objective, unless the penalty of the constraints is increased.

Like this penalty, many other hyperparameters that we tuned during the experiments could be changed, but they were not discussed as they are not generally dependent on the models used or the tasks, but we found that work better in almost all cases. Examples are tolerance for stopping criterion, number of epochs and learning rate of the ADAM optimizer used for parameter search, or the variance used for the initialization of parameters. However, changing these values within the limits of reason would not change the results, and could at most change the compromise between the elapsed time and the goodness of the performance.

Another subject of discussion could be the choice of the features passed to the MIA regressor to attack the privacy levels of the models, as discussed before many strategies could be applied, we decided for our examples to use loss because using our models it was enough to show how the unlearning processes worked, but there is the possibility to combine other features as logits, entropy, confidence and correctness of predictions.

We can conclude that our work can be considered satisfying as the main objective of creating a tool to generate controlled synthetic tabular data was reached. The main reasons that motivated this work were that collecting data for in our case Machine Unlearning training, or more generally for Machine Learning training, is often much expensive in terms of energy, time and money, and in this thesis it was shown that with the presented tool datasets of various characteristics and sizes can be generated easily, with almost total control capability and in a reasonable time.

7. Bibliography

www.ultralytics.com/glossary/adam-optimizer

Zhao, Kurmanji, Barbulescu, Triantafillou E., Triantafillou P. - What makes unlearning hard and what to do about it

Panagiotou, Roy, Ntoutsi - Synthetic Tabular Data Generation for Class Imbalance and Fairness: A Comparative Study

Fonseca, Bacao - Tabular and latent space synthetic data generation: a literature review

Reiter - Using CART to Generate Partially Synthetic, Public Use Microdata

Patki, Wedge, Veeramachaneni - The Synthetic data vault

Chawla, Bowyer, Hall, Kegelmeyer - SMOTE: Synthetic Minority Over-sampling Technique

Li, Xiong, Jiang - Differentially Private Synthesization of Multi-Dimensional Data using Copula Functions

Savelli, Giobergia, Baralis - Bad Teaching in Machine Unlearning with Similarity-based Sampling

Houssou, Augustin, Rappos, Bonvin, Robert-Nicoud - Generation and Simulation of Synthetic Datasets with Copulas

Wang, Sudalairaj, Henning, Greenewald, Srivastava - Post-processing Private Synthetic Data for Improving Utility on Selected Measures

Kodge, Saha, Roy - Deep Unlearning: Fast and Efficient Training-free Approach to Controlled Forgetting

Bu, Jin, Vinamuri, Ramakrishna, Chang, Cehver, Hong - Unlearning as multi-task optimization: A normalized gradient difference approach with an adaptive learning rate

Ding, Sharma, Chen, Xu, Ji - Understanding Fine-tuning in Approximate Unlearning: A Theoretical Perspective

Hong, Zou, Hu, Zeng, Wang, Yang - Dissecting Fine-Tuning Unlearning in Large Language Models

Geng, Li, Woisetschlaeger, Chen, Cai, Wang, Nakov, Jacobsen, Karray - A Comprehensive Survey of Machine Unlearning Techniques for Large Language Models