Machine Unlearning
CS772 Project Report

Gaurang Dangayach
Roll Number: 200373

Rahul Rustagi
Roll Number: 200756

Suryanshu Kumar Jaiswal
Roll Number: 201025

Udvas Basak
Roll Number: 201056

Ujjwal Kumar
Roll Number: 201059

1 Problem Description

Machine Unlearning is the problem of forgetting the knowledge of some training examples from an already learned model. In a Bayesian context, we want to remove the influence of some of the likelihood terms from the posterior, thereby mitigating the impact of specific training point. The methods developed in this approach are useful for eliminating outdated, irrelevant or stale data, de-biasing models or for rectifying inaccuracies.

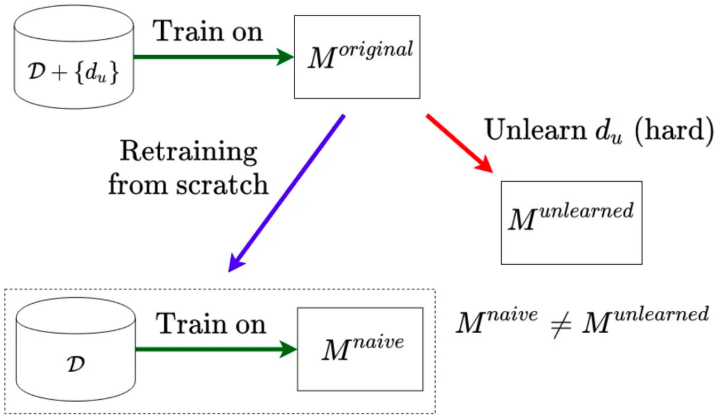

Figure 1: To remove the contributions of a data point, D_u, we can either use unlearning or retrain the model from scratch. *[3]*

When users request for the removal of their data, it is reasonable to assume that any contributions their data may have made to downstream models will also be erased. By doing this, the user's privacy is shielded from potential attackers who would try to access it by deducing private user data that was utilised to train the model.

But, what does it mean to remove a data points contributions? The main idea stands that training the model again on D_r (Remaining Dataset) would be computationally expensive.

We would be reviewing the algorithm implemented and its performance in the following sections:

2 Literature review and description of prior work on the problem

A framework to identify and nullify erroneous data: Tanno et al. (2022) [1] This was the main research paper on which our project is based and we have used the overall framework as suggested in this work while adding the finer details from other sources.

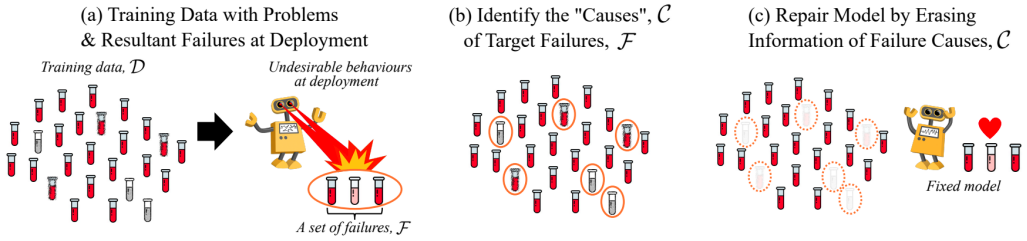

Figure 2: (a)Given the training data and failure set (b) We try to develop an algorithm that learns causes for failure (c) We repair the model by removing information of failure causes that affects the model parameters. *Reference: This figure is taken from Tanno et al. (2022) [1]*

Here, we want explore algorithms that try to identify the causes of failure from the training data and update the model by erasing the 'memories' of this harmful data.

Table 1: Notation and Definitions

Notation	Definition
D	The training dataset
D_e	The erased dataset ($D_e \subset D$)
D_r	The remaining dataset (i.e., $D_r \equiv D \backslash D_e$)
θ_D	The parameters of the model trained on D
θ_{D_r}	The parameters of the model (re)trained on D_r
Θ	The candidate set
$w(\theta)$	The weight of a candidate $\theta \in \Theta$

2.1 Step I: Cause Identification

Train & Test: We train a Bayesian model on the available training data (D) and then test it on the test data. We mark the test points on which the model gave bad answers as F called Failure set.

Update posterior: We then apply a continual learning method to obtain a new posterior

$$q'(\theta) = p(\theta|D, F)$$

This is done by fitting the model on the Failure set F.

Compute influences of training examples on F: We try to numerically quantify the amount of influence different training points have on the model which resulted in the undesirable behaviour on the Failure set on the time of deployment. We define such a function $r(z)$ and compute it $\forall Z \in D$

Find failure causes C: We can select the training examples which give the positive value of r(z) and we put them in one set C which is Failure Causes.

Algorithm 1 Model Repairment

Input: training data D; failure cases $\mathcal{F}$; approximate posterior $q(\theta) \approx p(\theta|D)$; likelihood $p(z|\theta)$
Output: failure causes $\mathcal{C}$, "repaired" posterior $q_{-\mathcal{C}}(\theta)$
Step I: Cause Identification
Update posterior: Apply a *continual learning* method to obtain $q_{+\mathcal{F}}(\theta) \propto p(\theta|D, \mathcal{F})$ by fitting the failure set $\mathcal{F}$
Compute influences of training examples on $\mathcal{F}$: Calculate $r(z)\ \forall z \in D$ (Eq. (9))
Find failure causes $\mathcal{C}$: Return the examples with positive influence, $\mathcal{C} \leftarrow \{z \in D : r(z) > 0\}$

2.2 Step II: Treatment Delete information of C:

Markov Chain Monte Carlo-Based Machine Unlearning: Nguyen et al. (2022) [2]
This the second research paper which we have taken up methods to implement the framework suggested in the first research work.

This paper focuses on solving supervised learning problems using Bayesian ML models which are used to capture the conditional probability

$$p(y|x, \theta)$$

The problem is to remove the effects of a set D_e of unwanted data (also referred to as erased dataset) in a model trained on $D \supset D_e$, by finding an approximation of the model parameters without involving the costly procedure of retraining the model on $D_r = D \backslash D_e$

The proposed solution is based on MCMC sampling method to perform this unlearning.

The basic formulation is as follows - Consider a discrete set Θ which we refer to as the candidate set of unlearned model parameters. This set Θ is constructed without the knowledge of the erased dataset D_e.

We would use MCMC methods to draw samples from the posterior distribution of Θ given D.

Therefore, the candidate set Θ is a set of samples drawn the distribution $p(\theta|D)$

Together with the candidate set Θ, we also store the values

$$h(\theta) = \log p(D|\theta) + \log p(\theta)$$

for all candidates $\theta \in \Theta$. We also define

$$g(\theta, D_e) = h(\theta) \log p(D_e|\theta)$$

We can make use of these functions to evaluate $\log p(\theta|D_r)$

$$\log p(\theta|D_r) = \log p(D_r|\theta) + \log p(\theta) - \log p(D_r) = g(\theta, D_e) \log p(D_r)$$

Recall that Θ is constructed as MCMC samples from the posterior distribution $p(\theta|D)$, so we can assign weight $w(\theta)$ to each candidate θ in Θ as follows -

$$w(\theta) = \frac{p(\theta|D_r)}{p(\theta|D)} = \frac{p(D_r|\theta)p(\theta)p(D)}{p(D|\theta)p(\theta)p(D_r)} = \frac{e^{g(\theta, D_e)}p(D)}{e^{h(\theta)}p(D_r)}$$

where $p(D)/p(D_r)$ is independent of θ, so it disappears after we normalize the weights for all $\theta \in \Theta$

We can use this weighted set Θ to approximate the posterior distribution $p(\theta|D_r)$.

3 Novelty of our work as compared to prior work

We have devised a new approach to implement cause identification [1] and then model treatment [2]. By using the algorithms and methods from the two different papers, we have achieved different accuracy

We in some sense combined the approaches of the two research works to formulate our final model. We used the framework of the first model and used MCMC in computing influences of training examples on F and also in performing the unlearning as suggested in the second paper.

Here is the final procedure we followed:

1. Train and test the Bayesian model and get the Failure set using usual procedures.
2. We train a new model on both the Training Data and the Failure set to get $q'() = p(|D, F)$ and then calculate $r(z)$

$$\hat{r}(z) = (E)_{p(\theta|D)}[\log p(z|\theta)] - (E)_{p(\theta|D,F)}[\log p(z|\theta)]$$

$$p(z|\theta) = p(y|x, \theta)$$

   These expectations are calculated using the MCMC samples.
3. The top K training data points with highest r(z) values are chosen to form Failure Causes C. This is our D_e also.
4. We generate samples from the posterior of the original model trained on training data using MCMC and calculate the corresponding weight $w(\theta)$ for each sample of θ using the formula proposed in the second research paper.
5. We calculated the weighted average of $p(y|x, \theta)$ over the samples of θ obtained from MCMC.

Finally, we compare the performance of the original model and the results obtained from the above procedure on D_r.

4 Description of tools/softwares used

To implement the above said pipeline, we used the following tools:

- **PyMC3**: A probabilistic programming library for Python that allows users to specify probabilistic models using an intuitive syntax and then perform Bayesian inference on these models.
  - Probabilistic Programming: PyMC3 allows you to express complex probabilistic models using a syntax similar to mathematical notation, making it easy to define and understand Bayesian models.
  - Automatic Differentiation: It leverages Theano's automatic differentiation capabilities to efficiently compute gradients required for Bayesian inference algorithms like Markov Chain Monte Carlo (MCMC) and Variational Inference (VI).
  - Model Checking and Diagnostics: PyMC3 provides built-in tools for model checking and diagnostics, allowing assessment of models and inference results.
- **Platform**: Google Collab
- **Python**: sklearn, numpy, pandas

5 Experimental results including description of data

We have provided the links for implementation of Model and Dataset that we used for this project. Please find them below.
Code Link, Dataset Link

Dataset Description:

We are using the diabetes dataset which is available on Kaggle. The data was originally collected by the National Institute of Diabetes and Digestive and Kidney Diseases from a set of females at least 21 years old and of Pima Indian Heritage.

The objective is to use the patient information to predict whether or no the patient has diabetes. There are 8 features (explanatory variables) and 1 label (response variable). This data collected from acutal patients and represents a task which might commonly be undertaken by a human doctor interested in identifying the patients most at risk for diabetes in order to recommend preventative measures.

Below are the observations collected after implementing the above mentioned algorithms on the dataset.

Experimental Results:

- The pipeline is coded and tested on the standard logistic regression modelling of diabetes data.
- It was observed that accuracy decreased when this pipeline was used. Because of that, Step 1 of the process was reobserved.
- A (0.8, 0.2) train-test split is created on the 768 datapoints(after median imputation), and 50% of the train labels are corrupted. Hence, X_train and corrupted_X_train is created.
- (Accuracy, F1 Score)$_{X\_train}$ = (73.38%, 0.55)
  (Accuracy, F1 Score)$_{corrupted\_X\_train}$ = (54.55%, 0.4262)
- Then $\hat{r}(z)$ is calculated, and all the samples with $\hat{r}(z) > 0$ are demarcated as $\mathcal{C}$. When cross-checked with the initially corrupted samples, only 60.58% of these corrupted samples were found in $\mathcal{C}$.
- Removing the influence of $\mathcal{C}$ by retraining gives (Accuracy, F1 Score)$_{filtered\_X\_train}$ = (70.13%, 0.50), which suggests that this model is better than a pure corruption, but worse than a fully cleaned model

6 Calculations

- Use the samples from the Trace Log of original model and calculated the weight of each sample using the formula :

$$w(\theta) = \frac{1}{\mathbb{P}(D_e)}$$

where

$$\mathbb{P}(D_e) = \prod P(y_i|x_i, \theta)$$

In case of Logistic Regression it turns out to be : logit = mean vector × data point vector

$$\mathbb{P}(y_i = 1|x_i, \theta) = \frac{1}{1 + exp(-logit)}$$

$$\mathbb{P}(y_i = 0|x_i, \theta) = 1 - \frac{1}{1 + exp(-logit)}$$

- We calculate the value of $\mathbb{P}(y_i|x_i, \theta)$ for all the samples using the same formula and take the weighted average to get the expected value of y_i
  In case of logistic regression, $\mathbb{P}(y_i = 1|x_i, \theta)$ gives us expected value of y_i. If this expected value comes out to be > 0.5, we assign the final value of $y_i = 1$ else 0.

7 Project Learnings

- The project gave us a hands-on-understanding of implementing basic Bayesian Machine Learning Models and procedures in code.
- This gave us an route into a fresh field of ML, Machine Unlearning
- This also gave us an idea of how many approximation techniques are actually employed, with exact examples of situations where each of them are most employable.

8 Possible Future Work

- In this project, we have implemented the algorithm only on Logistic Regression model. We can try to implement the algorithm on other models. However, during literature review we felt that output of Linear regression models is a bit uncertain (due to no formal expression of output), so we went ahead with logistic regression model. One can also extend this to classification models.
- Different Continual learning methods can be explored for directly fine-tuning the model and aiming for a more-efficient computation. Examples include using gradient ascent, or using the previous iteration posterior as prior to compute new posterior in the next iteration.
- Reciprocating the weight allocation relation for training unlearning model would yield importance metric corresponding for best examples to learn in case of a standard learning model. This can be explored more for efficient training and getting better accuracies.

9 Work Contribution

Below is roughly the contribution of each member in this project:

Member	Contribution
Gaurang Dangayach	Implementing Model - Treatment algorithms
Rahul Rustagi	Analysis of Model-Treatment Research Paper
Suryanshu Kumar Jaiswal	Failure Detection Paper Analysis
Udvas Basak	Paper Analysis and Failure Cause Identification Model Implementation
Ujjwal Kumar	Failure Detection Paper Analysis

References

[1] R. Tanno, M. F. Pradier, A. Nori, and Y. Li. Repairing neural networks by leaving the right past behind. ArXiv, abs/2207.04806, 2022.

[2] Q. P. Nguyen, R. Oikawa, D. M. Divakaran, M. C. Chan, and B. K. H. Low. Markov chain monte carlo-based machine unlearning: Unlearning what needs to be forgotten. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '22. ACM, May 2022.

[3] Fig1: Christopher Choquette, https://medium.com/@choquette.christopher/what-is-machine-unlearning-pt-1-933ff53dc9a6