Dissertation Titled

"CREDIT RISK ASSESSMENT FROM COMBINED BANK RECORDS USING FEDERATED LEARNING"

Submitted in partial fulfillment of the requirements of the degree of Bachelor of Technology

By

DEEP KAWA (151070044)
SUNAINA PUNYANI (151071006)
PRIYA NAYAK (151071016)
ARPITA KARKERA (151071041)

B. Tech. Computer Engineering
2018-2019

Under the guidance of
PROF. VARSHAPRIYA JYOTINAGAR

Department of Computer Engineering and Information Technology

Veermata Jijabai Technological Institute

H. R. Mahajani Marg, Matunga, Mumbai - 400 019

2018-2019

APPROVAL SHEET

The dissertation entitled **"CREDIT RISK ASSESSMENT FROM COMBINED BANK RECORDS USING FEDERATED LEARNING"** by Deep Kawa (151070044), Sunaina Punyani (151071006), Priya Nayak (151071016) and Arpita Karkera (151071041) is found to be satisfactory and is approved for the degree of Bachelor of Technology.

_____

Prof. Varshapriya Jyotinagar

Project Guide

Date: _____

Place: VJTI, Mumbai

CERTIFICATE

This is to certify that Deep Kawa (151070044), Sunaina Punyani (151071006), Priya Nayak (151071016) and Arpita Karkera (151071041), students of B. Tech. Computer Engineering, Veermata Jijabai Technological Institute (VJTI), Mumbai have successfully completed the Bachelor of Technology Dissertation entitled **"CREDIT RISK ASSESSMENT FROM COMBINED BANK RECORDS USING FEDERATED LEARNING"** under the guidance of Prof. Varshapriya Jyotinagar.

Prof. Varshapriya Jyotinagar
Project Guide

Examiner

Dr. M. M. Chandane
HoD, CE & IT

STATEMENT OF CANDIDATES

We state that work embodied in this Project entitled **"CREDIT RISK ASSESSMENT FROM COMBINED BANK RECORDS USING FEDERATED LEARNING"** forms out own contribution of work under the guidance Prof. Varshapriya Jyotinagar at the Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute. The report reflects the work done during the period of candidature but may include related preliminary material provided that it has not contributed to an award of previous degree. No part of this work has been used by us for the requirement of another degree except where explicitly stated in the body of the text and the attached statement.

Deep Kawa	Sunaina Punyani	Priya Nayak	Arpita Karkera
(151070044)	(151071006)	(151071016)	(151071041)

Date: _____

ACKNOWLEDGEMENT

We would like to thank all those people whose support and cooperation has been an invaluable asset during the course of this Project. We would also like to thank our guide Prof. Varshapriya Jyotinagar for guiding us throughout this project and giving it the present shape. It would have been impossible to complete the project without their support, valuable suggestions, criticism, encouragement and guidance.

We convey our gratitude also to Dr. M. M. Chandane, Head of Department for his motivation and providing various facilities, which helped us greatly in the whole process of this project.

We are also grateful for all other teaching and non-teaching staff members of the Computer Technology for directly or indirectly helping us for the completion of this project and the resources provided.

REPORT ORIENTATION

The report will comprise of four chapters with different content and scenarios providing the complete details about the project. The report is completed in such a way that it first provides the background knowledge about the project and then gives the thorough details about it. It also has methodology and the conclusion. The different chapters of the report are as follows.

Chapter 1: Background and Motivation

This chapter will provide introduction to the project and motivation for performing it. This chapter gives a review of the current credit risk assessment techniques and previous attempts, followed by description of approach, and problem statement.

Chapter 2: Literature Review

This chapter will provide literature studied for the project. The focus would be on the technology going to be used.

Chapter 3: System Analysis and Design

This chapter is basically divided into System analysis and System Design. In this chapter we provide system architecture and analyse it. We also focus on detail information of the data sets and we also have time line chart.

Chapter 4: Implementation and Comparison

This chapter describes the implementation and the results obtained. The method is also compared with previous techniques to see which is better.

Chapter 5: Conclusion

This chapter gives the conclusion and the issues, improvements and future scope.

ABSTRACT

Granting loans is the core business part of financial institutions. Revenue is generated by lending money at rates higher than the cost of money lent. This brings the institutions at risk from defaulting customers. To reduce the the credit risk, institutions assess credit risk of the prospective borrowers. Various subjective and quantitative factors are used to predict credit risks. The prevailing advancements in AI/ML has led to the advent of machine learning to generate effective predictive models by using customer data. Better models can be created by using data from multiple institutions. But sharing of data from multiple institutions puts the privacy of customers' data at risk and incurs huge communication costs. In this dissertation we are using federated learning to enable different financial institutions to collaboratively learn a shared credit risk prediction model while keeping the data with the respective institutions. Thus, eradicating the need for any centralisation of data. Another aim would be to compare this method and previous techniques with respect to their prediction, proficiency and accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This chapter provides the background and motivation for the work carried out in this project. The motivation of development of "Credit Risk Assessment from Combined Bank Records using Federated Learning". Previuos attempts in this field and the current scenario will be studied. Finally, the problem statement will be stated and proposed solution will be introduced for the same.

1.1 Background

Loan distribution is the core business part of financial institutions. Revenue is generated by lending money at rates higher than the cost of money lent. This brings the institution at risk from defaulting customers. Default is the failure to meet legal obligations of a loan, i.e., when a borrower fails to repay the loan. The biggest private default in history is Lehman Brothers, with over $600 billion when it filed for bankruptcy in 2008. In India, as of March 2018, the gross Non-Performing Assets (NPA), or bad loans, for Indian banks stood at Rs. 10.36 trillion.

In order to prevent possible bank capital losses and possible bankruptcy, an effective credit risk management is crucial for the success of financial institutions. A credit risk is the risk of default on a debt that may arise from a borrower failing to make required payments. Due to non- repayment of the loan the lender incurs losses. The loss may be complete or partial. In an efficient market, higher levels of credit risk is associated with higher borrowing costs.

To reduce the lender's credit risk, the lender may perform credit check on the prospective borrower. Credit risk assessment is the method by which one calculates the creditworthiness of an individual, business or organization. It helps banks to evaluate if a loan applicant can be defaulter at a later stage.

1.2 Previous Attempts

Traditionally, credit risk assessment was performed manually by risk analysts employed by institutions for this purpose. Various subjective and quantitative factors of a borrower are considered and using statistical and mathematical techniques a credit rating is generated. An increase in use of credit has led to large amount of data being collected. Data at this scale is difficult to process by humans.

With an effort to mechanize the analysis many algorithms were developed. The first researches into credit scoring were done by Fisher and Durand, who applied linear and quadratic discriminant analysis respectively to categorize credit applications as good or bad ones. Over the years a lot of research has gone into developing more efficient models. The models have usually depended on data collected from customers. Few researches and algorithms include domain knowledge in the analysis.

1.3 Current Scenario

A machine learning model can yield much better insights from the data than a human analyst. Presently, a lot of research has been done to develop models using machine learning. Many different training techniques have been incorporated and have shown to improve accuracy. The risk context predominantly determines which model to be used for the analysis. There is no prescriptive method entirely tied to a set of algorithms.

In India, currently, banks either use model trained by third party or train their own model on their own customer data for credit risk prediction. For training their models, third parties collect data from different financial institutions.

1.4 Motivation

Today, many banks/financial companies approve loans after a regressive process of verification and validation. Still, the number of loan default cases have only increased.

Although financial institutions have moved from manual approach to machine learning based approach for credit risk prediction, the results are still not very satisfactory. This is because the models used for prediction are trained only a specific set of past customers, often that single bank's customers. This makes the model biased and less general as it is trained on limited data

It is known that large amounts of data afford simple models much more power; 1 trillion data points make it more easier to detect outliers than can 10 data points. The underlying distribution also becomes clear with more data. Consequently, less data would require more sophisticated normalisations and transformation routines on the data before it can become useful. Generally, the more the amount of data present for training, the model would be better and the predictions would be accurate.

It would be desirable for financial institutions to be able to share their data to build better models, since modern machine learning models such as neural networks require huge amounts of data that a single institution might not be able to provide. However, there are a few challenges in this approach.

The collected data from all the banks would have to be stored at a centralised storage site. There are always privacy and security implications of aggregating any large volume of data. It becomes a target for attackers and criminals. Prevention of such activities involve constant vigilance of the data system. A single vulnerability in the system can cause a major and devastating data breach. There are agencies like Equifax, TransUnion, Experian which store the data centrally of all the banks but the possible vulnerabilities are limitless. There have been instances of data breaches in these organisations as well. Also, the banks would be reluctant to share their data with others because of business reasons.

We have proposed a method for creating shared prediction model which is essentially trained on data from across different banks without actually accessing the raw data.

1.5 Problem Statement

1.5.1 Problem

Financial institutions perform credit risk assessment of borrowers before granting loans to them to reduce the risk of default. This assessment calculates creditworthiness of the borrower by considering various subjective and quantitative factors. Banks have been exploring machine learning to create their prediction models recently but these models are trained only on their own customers' limited data.

The aim is to create better credit risk assessment models for loan default prediction by using data from different banks without accessing the raw data or collecting it. Another aim is to compare the results obtained with the previous methods.

1.5.2 Proposal

We propose a solution using federated learning, which eliminates the need to store the data at a centralised location or to share the data with third parties, thereby reducing the above concerns. The central server which holding the machine learning model would send the current model to the bank servers, which would then train the model on their respective data and return the change in gradients back to the server. The server, after accumulating all the gradient changes, averages them to modify the existing model and this procedure goes on till convergence is reached. The model obtained is far more superior than the individual models as it is effectively trained on large data.

1.6 Scope

Usage of more data from all banks will lead to a better model, with better prediction capabilities as compared to the current models used by the banks on their own data as the model would have seen more instances and hence is capable of more accurate classification based on the provided features.

Due to the superiority and more capabilities acquired by the credit risk analysis model, its usage benefits all the banks, without the requirement to expose their data, thus providing a win-win situation. It gives the banks equal footing to fight financial crimes and prevents the scenario of data dominance.

Our solution is advantageous to the new and young customers who don't have a credit score as it takes into account other factors like age, profession, income, debt, assets as well along with credit score. This reduces the dependency on the credit score.

CHAPTER 2

LITERATURE SURVEY

2.1 Generic Credit Risk Analysis used in Banks

2.1.1 Credit Risk

A credit risk is the risk of default on a debt that may arise from a borrower failing to make required payments. The risk is that of the lender and includes lost principal and interest, disruption to cash flows, and increased collection costs. To reduce the lender's credit risk, the lender may perform a credit check on the prospective borrower, may require the borrower to take out appropriate insurance, such as mortgage insurance, or seek security over some assets of the borrower or a guarantee from a third party. [2]

A credit risk can be of the following types:

- Credit Default Risk - The risk of loss arising from a debtor being unlikely to pay its loan obligations in full or the debtor is more than 90 days past due on any material credit obligation.
- Concentration Risk - The risk associated with any single exposure or group of exposures with the potential to produce large enough losses to threaten a bank's core operations.
- Country Risk - The risk of loss arising from a sovereign state freezing foreign currency payments (transfer/conversion risk) or when it defaults on its obligations (sovereign risk).

2.1.2 Current Loan Distribution Practices

To manage credit risks, banks use the credit analysis procedures which gives a rough estimation of a customers worthiness, how likely is it that a customer will repay the loan. The objective of credit analysis is to look at both the borrower and the lending facility being proposed and to assign a risk rating. The risk rating is derived by estimating the probability of default by the borrower at a given confidence level over the life of the facility, and by estimating the amount of

loss that the lender would suffer in the event of default. Credit analysis involves a wide variety of financial analysis techniques, including ratio and trend analysis as well as the creation of projections and a detailed analysis of cash flows. Credit analysis also includes an examination of collateral and other sources of repayment as well as credit history and management ability. Analysts attempt to predict the probability that a borrower will default on its debts, and also the severity of losses in the event of default. [3]

Credit score is one way of credit analysis. A credit score is a statistic to determine a person's ability to pay back a loan. It normally ranges between 300-850 with 300 being the absolute lowest and 850 being the best possible score. The higher the credit score number a person has, the easier it is qualify for a loan. Credit score is based on information pulled from credit report. Payment history equates to 35% of the score. The second factor is how much an individual owes and this accounts for 30% of the score. Credit history makes up 15% of the score and the last application for credit accounts for 10%. The score focuses on three main factors:

- Recency
- Frequency
- Severity

A detailed view of the credit score and its calculations can be found in [4].

CIBIL Score is one of the credit scores, and so are Equifax, Highmark and Experian scores in India. Any numerical expression to determine the credit-worthiness of an individual, assigned by a Credit Bureau can be called Credit Score.

CIBIL Credit Information Reports

A CIBIL report is a highly valued credit report that helps banks and financial institutions analyze a person's credit worthiness based on his/her repayment discipline and borrowing history. CIBIL Credit Information Report is divided into 6 sections: [14]

- CIBIL TransUnion score
- Personal Information
- Contact Information

- Employment Information
- Account Information
- Enquiry Information

2.1.3 Credit Reporting Agencies

Credit reporting agencies like Equifax, TransUnion, etc. maintain files on millions of borrowers. These files are shared to the credit bureaus by the affiliated banks. Lenders and other businesses, banks as well, use the information in the credit reports generated by these bureaus to evaluate an application for credit, loan, insurance, etc.

2.1.4 Third Parties and Associated Risks

Credit Bureaus

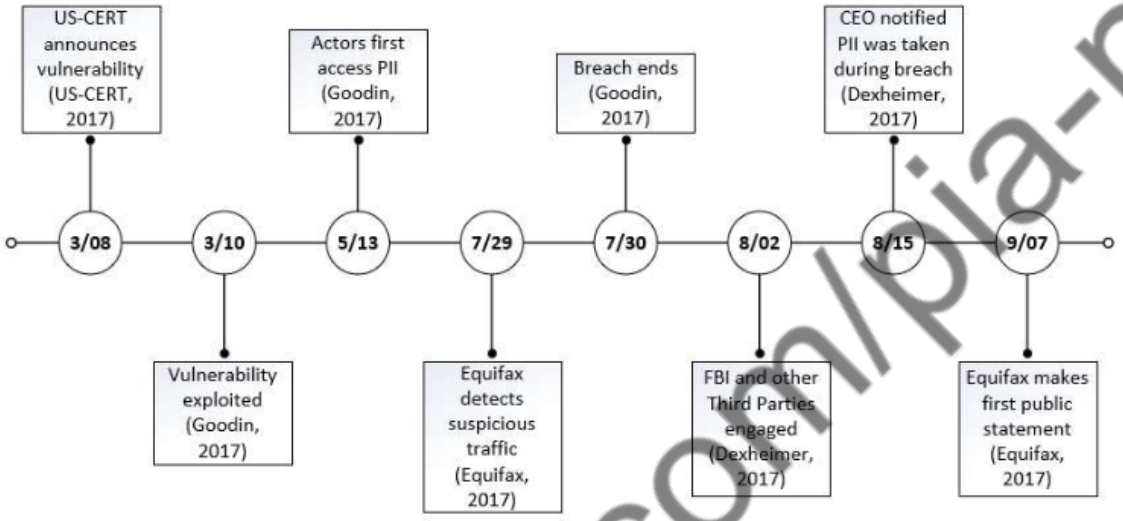
A credit bureau is an agency that collects and researches individual credit information and sells it for a fee to creditors so they can make a decision on granting loans. The three main credit bureaus in the United States are Equifax, Experian and TransUnion, though there are many smaller companies that provide similar services. The primary purpose of credit bureaus is to ensure that creditors have the information they need to make lending decisions. Typical clients for a credit bureau include banks, mortgage lenders, credit card companies and other financing companies. Credit bureaus are not responsible for deciding whether or not an individual should have credit extended to them; they merely collect and synthesize information about an individual's credit score and give that information to lending institutions. [5]

Equifax Data Breach

Data breach is a major threat to the third party data aggregators like credit bureaus. One such major incident was the Equifax Data Breach case in 2017.

The Equifax case is selected because it not only happened recently but also is a massive breach of sensitive personal and financial information including social security numbers, birth dates and addresses, and in some cases driver license numbers, credit card information and

financial dispute documents of over 140 million U.S. consumers (or over 44% of U.S. population). Equifax is a global information solutions company that uses unique data, innovative analytics, technology and industry expertise to power organizations and individuals around the world by transforming knowledge into insights that help make more informed business and personal decisions.



Soon after Equifax's initial press release of the data breach, its stock value closed down about 14 percent and the cost estimates for the company by the breach would be in hundreds of millions of dollars. In addition, Equifax was hit with 23 class-action lawsuits filed through the weekend after its initial announcement of the data breach.

A detailed analysis of the case can be found in [6].

2.2 Credit Risk assessment using Machine Learning

2.2.1 Multistage Neural network Ensemble Learning Approach

In this study, a multistage neural network ensemble learning model is proposed to evaluate credit risk at the measurement level [1]. The proposed model consists of six stages. In the first stage, a bagging sampling approach is used to generate different training data subsets especially for data shortage. In the second stage, the different neural network models are created with different training subsets obtained from the previous stage. In the third stage, the generated neural

network models are trained with different training data-sets and accordingly the classification score and reliability value of neural classifier can be obtained. In the fourth stage, a decorrelation maximization algorithm is used to select the appropriate ensemble members. In the fifth stage, the reliability values of the selected neural network models (i.e., ensemble members) are scaled into a unit interval by logistic transformation. In the final stage, the selected neural network ensemble members are fused to obtain final classification result by means of reliability measurement. For illustration, two publicly available credit data-sets are used to verify the effectiveness of the proposed multistage neural network ensemble model.

2.2.2 Credit Risk Analysis and Prediction Modelling of Bank Loans Using R

Credit Risk assessment is a crucial issue faced by Banks nowadays which helps them to evaluate if a loan applicant can be a defaulter at a later stage so that they can go ahead and grant the loan or not. This helps the banks to minimize the possible losses and can increase the volume of credits. The result of this credit risk assessment will be the prediction of Probability of Default (PD) of an applicant. Hence, it becomes important to build a model that will consider the various aspects of the applicant and produces an assessment of the Probability of Default of the applicant. This parameter PD, help the bank to make decision if they can offer the loan to the applicant or not. In such scenario the data being analysed is huge and complex and using data mining techniques to obtain the result is the most suitable option provided its efficient analytical methodology that finds useful knowledge. There are many such work has been done previously, but they have not explored the use of the features available in R package. R Package is an excellent statistical and data mining tool that can handle any volume of structured as well as unstructured data and provide the results in a fast manner and presents the results in both text and graphical manners. This enables the decision maker to make better predictions and analysis of the findings. The aim of this work is to propose a data mining framework using R for predicting PD for the new loan applicants of a Bank. The data used for analysis contains many inconsistencies like missing values, outliers and inconsistencies and they have to be handled before being used to build the model. Only few of the customer parameters really contribute to the prediction of the defaulter. So, those parameters or features need to be identified before a model is applied. To classify if the applicant is a defaulter or not, the best data mining approach

is the classification modelling using Decision Tree. Its implementation details can be studied in [11]

2.2.3 Hybrid Support Vector Machine Ensemble Model for Credit Scoring

Credit risk is the most challenging risk to which financial institution are exposed. Credit scoring is the main analytical technique for credit risk assessment. In this paper a hybrid model for credit scoring is designed which applies ensemble learning for credit granting decisions. The hybrid model combines clustering and classification techniques. Ten Support Vector Machine(SVM) classifiers are utilized as the members of ensemble model. Since even a small improvement in credit scoring accuracy causes significant loss reduction, then the application of ensemble in hybrid model leads to better performance of classification. A real data-set is used to test the model performance.The test results shows that proposed hybrid SVM ensemble has better classification accuracy when compared with other methods.the detailed description of this hybrid SVM model is given in [12].

2.3 Federated Learning

Standard machine learning approaches require centralizing the training data on one machine. Google has built one of the most secure and robust cloud infrastructures for the processing of this data to make services better. But the increased importance of privacy of data and the ongoing efforts for it has led it to introduce an additional approach: Federated Learning.

Federated Learning enables devices to collaboratively learn a shared prediction model while keeping all the training data on the device itself, decoupling the ability to do machine learning from the need to store the data in the cloud. This approach brings the model training to the devices.

The model gets downloaded from the server to the devices, where it is trained on the data on the devices and device summarizes the changes as a small focused update.This update is sent back to the server using encrypted communication, where it is immediately averaged with other user updates to improve the shared model. [7]

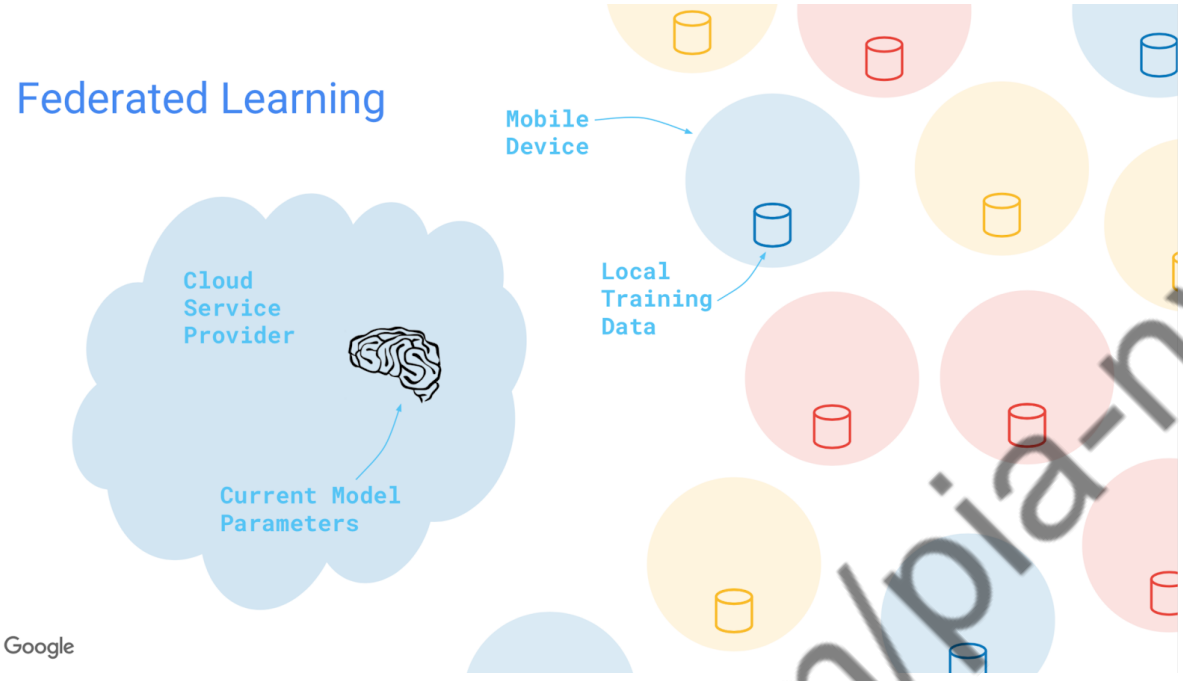


Figure 2.1: Components of Federated System (Source : Google AI Blog)

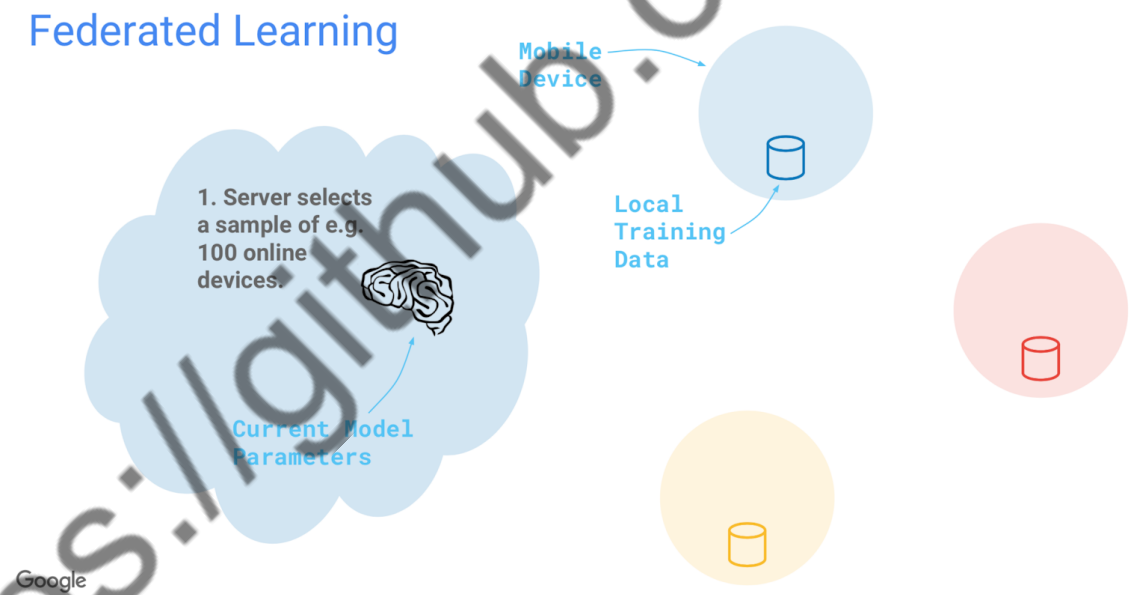


Figure 2.2: Server selects sample of online devices. (Source : Google AI Blog)

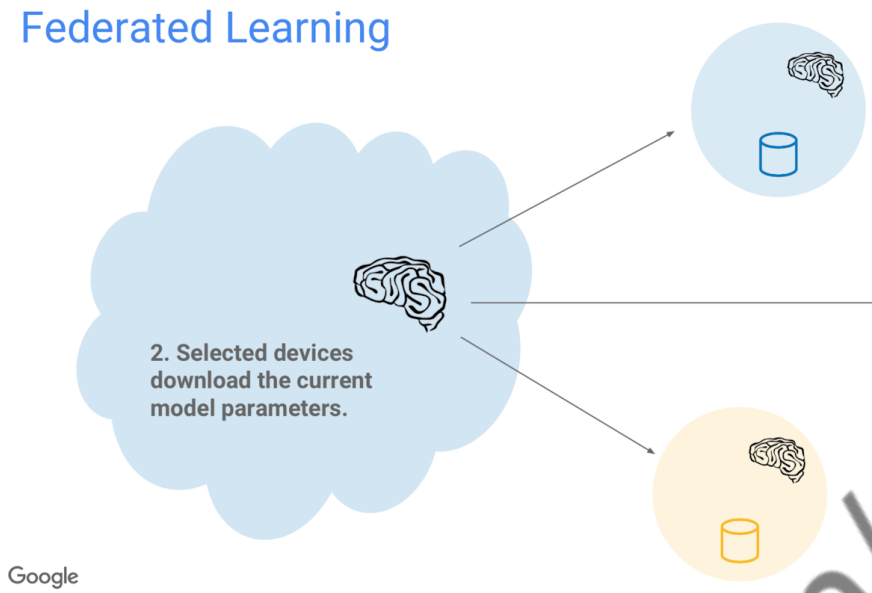


Figure 2.3: Selected devices download the current model parameters (Source : Google AI Blog)

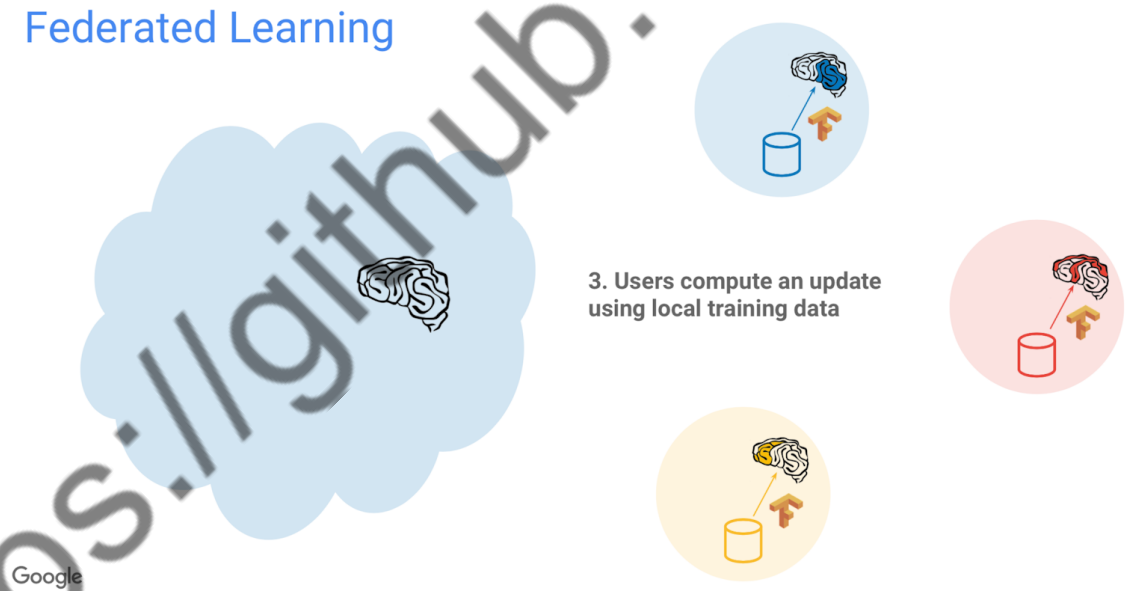


Figure 2.4: Users compute an update using local training data (Source : Google AI Blog)

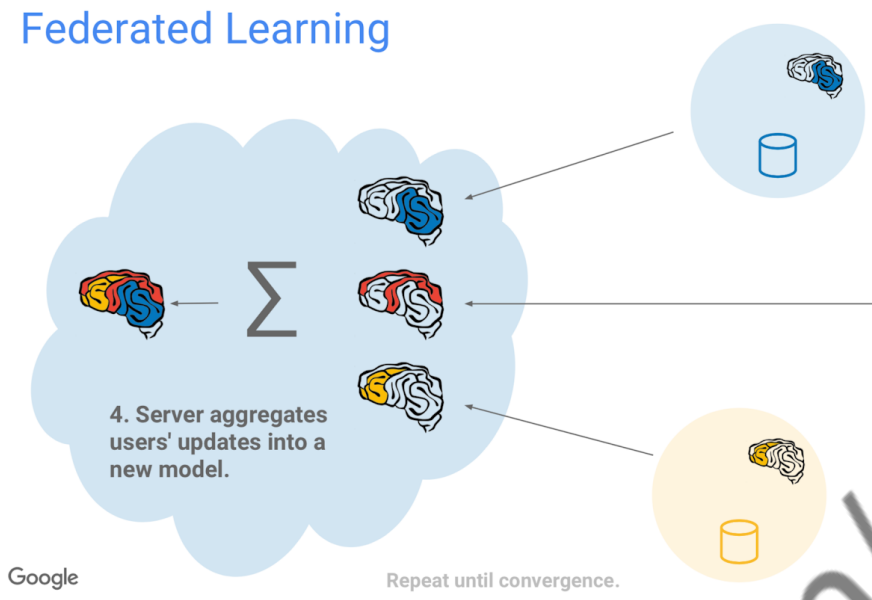


Figure 2.5: Server aggregates users update into a new model (Source : Google AI Blog)

2.3.1 Communication Requirements for Federated Learning

- Structured Updates

  This type of communication efficient update restricts the update to have a pre-specified structure. We train directly the updates of this structure, as opposed to approximating/sketching general updates with an object of specific structure.

  - Low Rank
  - Random Mask

- Sketched Updates

  This type first computes the full H during local training without any constraints, and then approximates, or encodes the update in a compressed form before sending to the server. Server decodes updates before aggregation.

  - Sub Sampling
  - Probablistic Quantization

- Federated Averaging Algorithm

  Select a C- fraction of clients on each round, and compute the gradient of the loss over all the data held by these clients. In each iteration, the server sends a model to C-fraction of clients. Each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

A brief analysis of the algorithms and techniques used for compression for safe transmission of updates to the server can be found in [8], [9].

2.4 Privacy and Security Requirements for Federated Learning

2.4.1 Secure Aggregation

Secure aggregation is a class of secure multiparty computation algorithms wherein a group of mutually distrustful parties u belonging to U each hold a private value 'x' and collaborate to form an aggregate value, without revealing to one another any information about their private value except what is learn able from the aggregate value itself. Secure aggregation is used to protect privacy of gradients of each user.

Although the updates on the user data are ephemeral and contains less information than the actual raw data, there has been evidence that a trained neural network parameters sometimes allow reconstruction of training examples. If the input 'x' is a one-hot vocabulary length vector encoding the most recently typed word, a common neural network architecture will contain

at least one parameter for each word such that its derivative is non-zero when 'x' encodes that word. Thus the set of recently typed words would be revealed by inspecting the non-zero entries of the partial derivative. The server does not need to inspect any individual user's update, it requires only the sum. Using secure aggregation ensures that server learns only the number of users that wrote the word but not explicitly which users. Secure aggregation protocol operates on high-dimensional vectors, is communication-efficient, robust, provides strongest possible security under the constraints of server mediated, unauthenticated network model.

Security requires that server learns nothing other than what is infer-able from the average of the updates and the user learns nothing. Three models are considered in this paper. It is assumed that all users follow the protocol honestly, but server may attempt to learn extra information in different ways:

- Server is honest but curious, that is it follows the protocol honestly but it tries to learn as much as possible from the messages it receives from the users.
- The server can lie to user about which other users have dropped out, including reporting dropouts inconsistently among different users.
- The server can lie about who dropped out and also access the private memory of some limited number of users.

The protocols used for this purpose are:n

- Masking with one time pads
- Dropped user recovery using secret sharing
- Double Masking to Thwart a Malicious Server
- Exchanging Secrets Efficiently
- Minimizing Trust in Practice

The protocols can be studied in detail in [10].

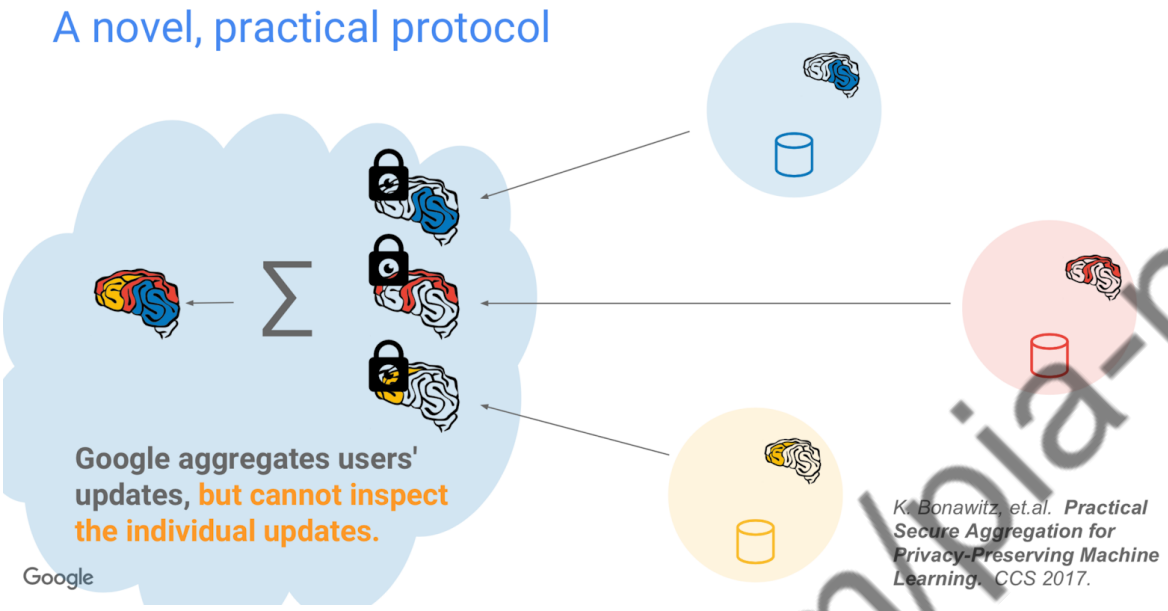


Figure 2.6: Secure Aggregation (Source : Google AI Blog)

2.4.2 Secure Multiparty Aggregation with Differential Privacy

Data privacy and security issues arise frequently and increasingly in data surveillance systems. An important subject is how to protect the privacy of the data subjects, especially when the data aggregator is untrusted or not present. Among all secure computation schemes three representative groups that are evaluated are secret sharing, homomorphic encryption and perturbation based. They represent different approaches of ensuring security splitting data into shares(secret sharing), encryption(homomorphic encryption) and perturbing data with significant noise(perturbation-based). Different methodologies for the implementation of these schemes can be studied from [13]

Secret Sharing

Secret sharing schemes split a secret into multiple shares such that at least t shares are required to reconstruct the secret. Additionally, any set of fewer than t shares disclose nothing about the secret. The value of t also defines the minimal number of colluding nodes necessary to breach the security of computations. Shares are distributed among participants, and each node receives a few (usually one) shares.

Perturbation-based Protocol

Perturbation-based protocols are an efficient alternative for encryption-based protocols. They usually require certain topology of node connections. For example, in a secure sum protocol, nodes are connected in a ring. The main idea behind these protocols is to perturb input data by adding some random noise, such that they become meaningless for an attacker, and then perform computations on the noisy data. This approach obfuscates all intermediate results, but is vulnerable to colluding nodes. In addition, it has low level of fault tolerance. Fault of any node requires rerunning the protocol by remaining active nodes. As an example of perturbation-based protocols, we consider the secure sum protocol. In this protocol all nodes are connected in a ring. Each node generates random noise, which is added to its private input. The starting node sends its perturbed value to its successor, which adds its own perturbed value, and passes it further. At the end of the first phase, the starting node gets the sum of perturbed values from all nodes. In the second phase, each node removes its noise from the perturbed sum, which, at the end, reveals the final sum. Unfortunately, if two neighbors of a node collude, then they can easily compute the perturbation, and the participated value of that node. Many enhancements have been introduced to this security scheme to increase its collusion resistance, e.g., shuffling node positions in the ring, and dividing computations into multiple rounds.

2.4.3 Homomorphic Encryption

Homomorphic encryption is a form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext.

Homomorphic encryption can be used for privacy-preserving outsourced storage and computation. This allows data to be encrypted and out-sourced to commercial cloud environments for processing, all while encrypted. Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted. Homomorphic encryption can be viewed as an extension of either symmetric-key or public-key cryptography. Homomorphic refers to homomorphism in algebra: the encryption and decryption functions can be thought as homomorphisms between plaintext and ciphertext spaces. Homomorphic encryption includes

multiple types of encryption schemes that can perform different classes of computations over encrypted data. Some common types of homomorphic encryption are partially homomorphic, somewhat homomorphic, leveled fully homomorphic, and fully homomorphic encryption.

An encryption scheme is homomorphic if it allows computations to be carried out on ciphertext. Formally, for a given homomorphic encryption function E with respect to a function f, the encrypted result of f can be obtained by computing a function g over encrypted values of x1.... xn,i.e.,

E(f(x1.....xn))=g(E(x1),.....E(xn))

In a distributed settings each party encrypts its data, and sends them to a single party, which computes the function g on them. Fully homomorphic schemes allow secure multiparty computation of any function f. The price for such flexibility in choosing f is high complexity. However, a few partially homomorphic schemes are efficient enough to achieve our security and performance goals, e.g., multiplicatively homomorphic ElGamal and RSA schemes.

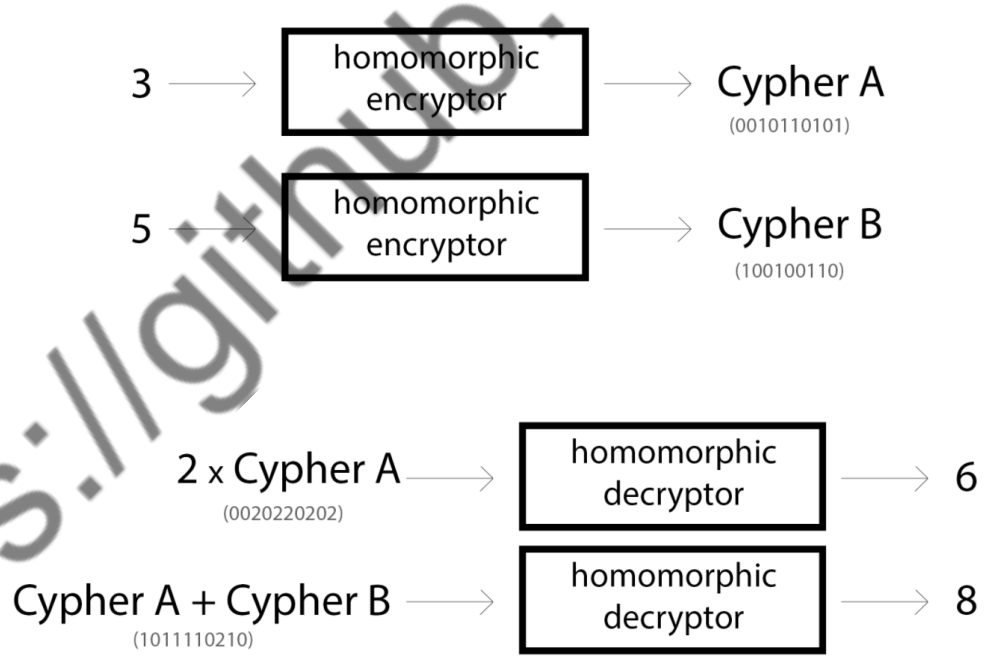


Figure 2.7: Homomorphic Encryption(Source : Google)

2.4.4 Differential Privacy

Differential privacy in deep learning is concerned with preserving privacy of individual data points. When a model is learnt in the conventional way, its parameters reveal information about data learnt during training. To avoid this, Differential Privacy is used, owing to which, the model doesn't reveal which data points were used during training. Ideally, the parameters of trained machine-learning models should encode general patterns rather than facts about specific training examples. To ensure this, and to give strong privacy guarantees when the training data is sensitive, it is possible to use techniques based on the theory of differential privacy. In particular, when training on users data, those techniques offer strong mathematical guarantees that models do not learn or remember the details about any specific user.

For differential privacy, a randomized mechanism is used, which consists of two steps: At the beginning of each communication round, a random subset of clients is chosen to contribute. Only these clients receive the central model and share their updates. Then, a Gaussian mechanism is used to distort the average of updates before allocating the new central model. This is done to hide a single client's contribution within the aggregation and thus within the entire decentralized learning procedure.

DP-SGD and PATE are two different ways to achieve the same goal of privacy-preserving machine learning. DP-SGD makes less assumptions about the ML task than PATE, but this comes at the expense of making modifications to the training algorithm. Indeed, DP-SGD is a modification of the stochastic gradient descent algorithm, which is the basis for many optimizers that are popular in machine learning. Models trained with DP-SGD have provable privacy guarantees expressed in terms of differential privacy.

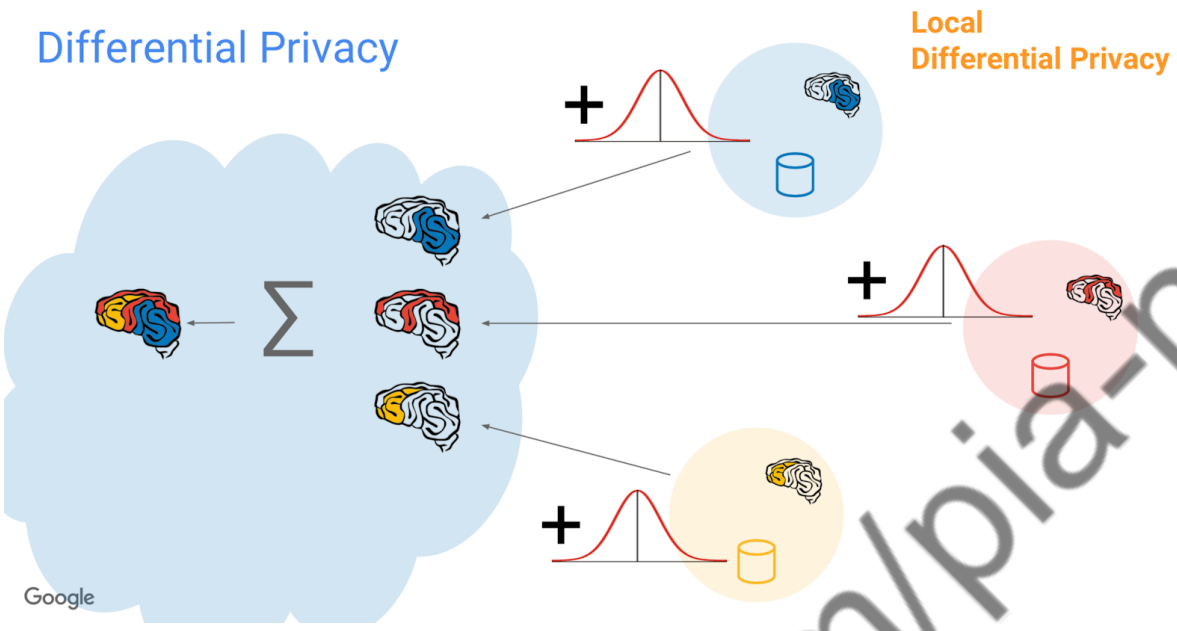


Figure 2.8: Differential Privacy (Source : Google AI Blog)

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 System Architecture

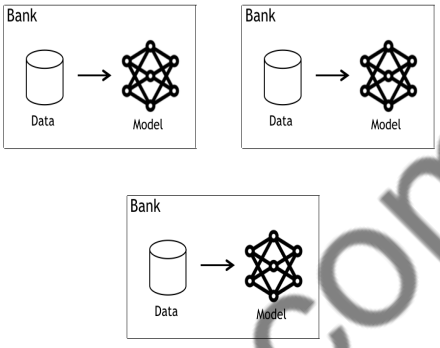


Figure 3.1: Current Machine Learning Systems

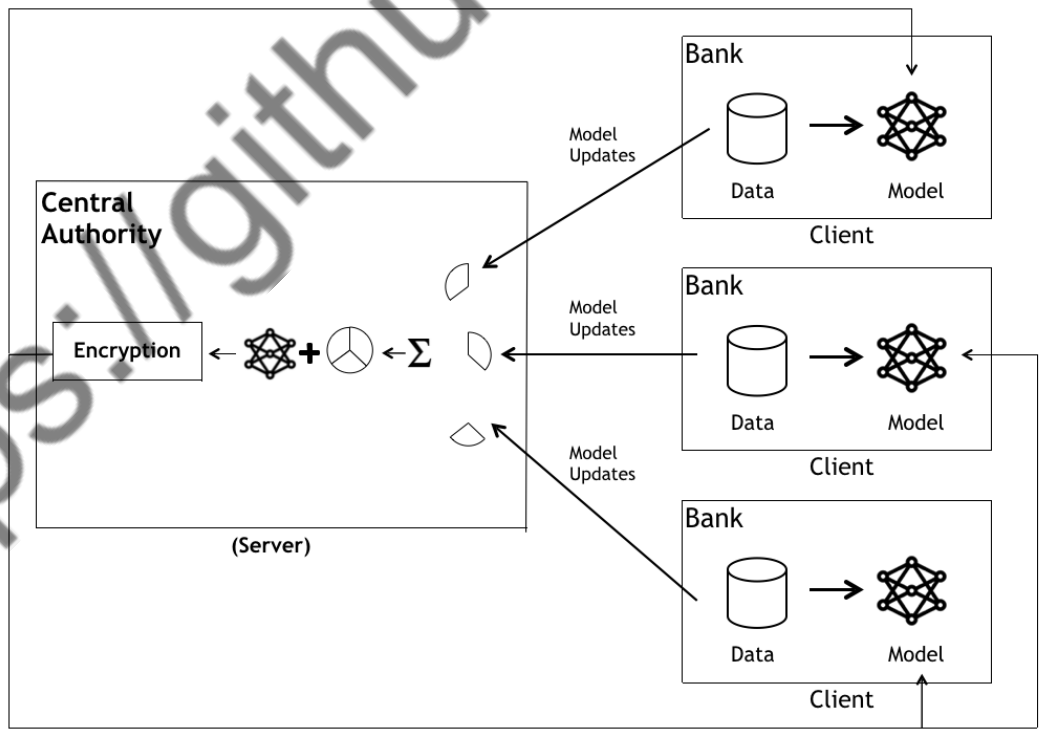


Figure 3.2: Proposed System Architecture

3.1.1 System Description

The system consists of a central authority server like Reserve Bank of India. The other participating servers are of the banks or any similar financial institution disbursing loans. The scope of the project demands all the servers need to be connected on the same network. For application on a larger scale, either the system could be connected by Internet (it would be secure because of Diffie Hellman and Homomorphic Encryption used) or Virtual Private Network. The multithread connection ensures that the connection between authority server and participating server persists along with the model training on the participants. The participants have to be authenticated by the authority server before the federation process starts. The system waits for the connection from the specified threshold number of clients.

3.1.2 Working

Loan distribution is the core business part of many banks. Banks face risk from customers who default. Currently banks use structural method (capturing economics) and empirical method (learning from data) to predict loan defaults. Banks use their own past customer data to create models using machine learning. Better models can be created by using more data, i.e., data from different banks. Centralization of data is not a feasible solution because of security, business reasons, and communication/storage costs. The proposed solution is to enable the banks to collaboratively learn a shared prediction model while keeping all the training data with the respective banks, decoupling the ability to do machine learning from the need to store the data in the cloud. This is achieved using federated learning. Our proposed system consists of central authority and the participating client banks with their respective data. The central authority will send a model (initially initialised with some values) to all the participating client banks. The client banks will individually download the model from the central authority and then train it on their own data. After this training step, the changes to the model will be summarized as a small focused update. Only this update will be sent back to the central authority, where the updates received from all client banks will be aggregated and added to update the shared model. This shared model will again be sent to client banks and the cycle will repeat till convergence. The resulting shared model will be a far superior model trained on data across different banks. To protect the model from individual client banks, the model will be homomorphically encrypted

during the whole process. Since the updates sent by client banks to the central authority can itself reveal information about the nature of data to the central authority, the central authority will have access to the aggregate of the updates from the clients but not to the individual update itself. This will be achieved using Secure Multiparty Aggregation. Differential Privacy is used to hide from the central authority, if a particular client bank has participated in the machine learning model or not, thus ensuring the privacy of the client banks. All the customer data required for training will remain with the client banks and no data or individual updates will be stored at the central server, eliminating a need for data collection and centralized storage. The consequent model on the central server will be more accurate, owing to the higher amount of data used in training. This model can be utilized by all the participating client banks creating a win-win situation of being able to avail the benefits of a better model without having to compromise the privacy of the data. The proposed solution has the following advantages : [5]

- There is no need to use complicated technology for collection, storage and ensuring the security of data as there is no collection of data at a central location
- The data is credible and trusted as it comes from verified participating banks.
- Since the customer data is sensitive, banks are usually reluctant to share data with other parties. In this approach, the data doesn't leave the bank and hence privacy is ensured.

CHAPTER 4

IMPLEMENTATION AND COMPARISON

Standard machine learning models require the data to be centralised in a datacenter or on one machine. Federated learning decouples the ability to do machine learning from the need to store data in the cloud. It enables devices to learn a shared prediction model by keeping the data on the devices. It brings the model to the data (device) rather than data going to the model. In the loan assessment scenarios, traditional machine learning model would require the data from all the financial institutions to be collected centrally. This might not be a feasible solution as the financial institutions would be reluctant to share their data because of business reasons, security and management issues of training data at the centers as well as the difficulty of transfer of huge amounts of data over unreliable channels. Using federated learning, it is possible to train a shared credit assessment model keeping all the data on the client bank server itself. Federated learning differs from the standard machine learning models in that the data used is not independent and identically distributed. In the proposed system, a central authorised institution initialises and keeps track of the latest model. The affiliated financial institutions download the current version of the model and train their respective data on this model, uploading back the updates or changes in updates. The central authority gathers and averages these updates from the affiliated financial institutions to update the main model and in the next round, this updated model is downloaded by the institutions. This process goes on till convergence.

4.1 Data

For this paper, we are using the loan data issued through 2007-2015, from Kaggle. The data set includes the current loan status and latest payment information. The training data target variable is a boolean variable whether or not the customer defaulted within the specified period of time. The testing data target variable will also be boolean, whether or not to grant the specified amount of loan to the customer. The data consists of approximately 12,70,000

data items and each data item has 20 features. As the data from all the financial institutions are not independent and identically distributed, the above data had to be separated into 3 parts in a non i.i.d way. The data is randomly divided within the 3 client banks, each client bank holds approximately 4,00,000 data items. The dataset used contains some key features including purpose for availing loan, installment, annual income, debt to income ratio, credit score, revolving balance, revolving utilities, inquiry in last 12 months, delinquency in last 2 years, public records, address state, employment length, type of home ownership, number of months since last delinquency, term, charge off within 12 months, loan amount, open accounts, total accounts. In total, we have a training dataset of approximately 12,70,000 data items and after the non i.i.d distribution, each set contains approximately 4,00,000 lakh data items. Each set will then be stored on the computers simulating the affiliated financial institutions. 10 percent of the original dataset has been taken as testing set.

Table 4.1: Data Features

Data Features	
Feature	Description
Purpose	Purpose of borrowing stated by borrower
Interest Rate	Interest rate on the loan
Installments	Monthly Payments owed by the borrower
Annual Income	Annual Income of the borrower
Debt to Income ratio	Ratio calculated using borrower's total monthly debt payments on the total debt obligations excluding mortgage and the requested loans divided by borrowers self-reported monthly income
Revolving Balance	total credit revolving balance
Revolving Utility	The amount of credit the borrower is using related to all the available revolving credit
Inquiries in last 12 months	Number of credit inquiries in last 12 months
Delinquency last 2 years	Number of 30+ days past due delinquency in borrowers credit file for past 2 years
Public Records	Number of Public Records
Address(State)	Location of the borrower
Employment Length	Employment Length in Years
Home Ownership	Ownership Status of home provided by borrower
Months since Last Delinquency	Number of months since the borrower's last delinquency
Term	Number of Payments on the loan
ChargeOff within 12 months	Number of chargeoff within 12 months
Loan Amount	Amount to be Borrowed

4.2 Privacy

4.2.1 Secure Aggregation

For the federated rounds to proceed further, the server needs the average of the updates from the participating financial institutions. In the federated setting described in Introduction, although each update is ephemeral and contains less information than raw data, there is a possibility that the server might find out about the data distribution of the participants by examining the individual updates. There also exists evidence that a trained neural network parameters sometimes allow reconstruction of training example. [Practical secure aggregation for federated learning on user-held data] Secure aggregation ensures that no individual updates are accessible to the server, only the average of the updates is. In applying federated learning on the device data, there is a chance that the device might get disconnected during the training round. But our idea is to use federated learning on the data present on the servers of financial institutions. The probability of servers getting disconnected between training rounds is very less and hence we assume that all the participating financial institution servers are present throughout the training round.

Protocol Used in this case: Masking with one-time pads using diffie hellman We are using pyDHE module for creation of diffie hellman keys between every pair of participating clients. The public keys P,G generated by the module depends on the seed specified. We use the same seed for every pair, which means only the private keys of every participant differs. Each participant generates a key using the public keys and their individual private key -

$$x = G^a mod P$$

and sends it to all other participants via the server. Since only the key generated is known to the server and not the private keys of the participants, server is incapable of deciphering the private keys. After the key exchange, if there are N participating financial institutions, each participant should have N - 1 keys. Randomized encrypted $s_{u,v}$ are created for every pair of participants and sent to the participants via the server. This encryption is done with AES using the Diffie Hellman key generated between the pairs of participants. We have used pyaes

module for simulating AES. The server has a dictionary of all the encrypted $s_{u,v}$ and send the appropriate $s_{u,v}$ to the participants. After the participants acquire the $s_{u,v}$ from other participants and decrypt them using the pairwise diffie hellman shared key, they subtract it from their own $s_{u,v}$ to create perturbations.

$$p_{u,v} = s_{v,u} - s_{v,u}$$

where,

$$p_{u,v} = -p_{v,u}(modR)$$

Here $s_{u,v}$ is a vector sampled uniformly from [0, R)k sampled by participant u for participant v. The dimensions of $s_{u,v}$ depend on the dimensions of the updates generated

$$y_{\mathrm{u}} = x_{\mathrm{u}} + \sum_{v \epsilon U} p_{\mathrm{u,v}}$$

is sent to the server by the participants where x u is the update from the training at the participant server u. At the server, taking the sum of all the received updates would cancel all the perturbations giving the sum of all updates to the server.

$$\bar{x} = \sum_{u \epsilon U} x_{\mathrm{u}} + \sum_{u \epsilon U} \sum_{v \epsilon U} p_{\mathrm{u,v}} = \sum_{u \epsilon U} x_{\mathrm{u}} + \sum_{u \epsilon U} \sum_{v \epsilon U} s_{\mathrm{u,v}} - \sum_{u \epsilon U} \sum_{v \epsilon U} s_{\mathrm{v,u}} = \sum_{u \epsilon U} x_{\mathrm{u}}(modR)$$

4.2.2 Differential Privacy

The possibility of tracing back individual clients contribution to the model must be prevented. The differential attack could be performed by any party contributing in the federated learning process. In these kinds of attacks, the contribution from a client during the training process and also the information about their dataset can be revealed through analysis of distributed parameters [15]. Basically, a model learnt with differential privacy is not affected by a single training example or a small set of training examples in the dataset. We use the Tensorflow differential privacy module for client side differential privacy. Tensorflow Privacy is a Python library that includes implementations of TensorFlow optimizers for training machine learning models with differential privacy. The module relies on Differentially Private Stochastic Gradient Descent.

This requires modifications to the training algorithms. The modifications required bounds the sensitivity of the gradients i.e how much each individual training point can influence the gradient computation - gradient clipping and to randomize the behaviour of the algorithm so as to make it impossible to find if a particular point participated in the training process by comparing the updates with or without the particular point in training set - noising.

4.2.3 Homomorphic encryption

The encryption technique which allows third-parties to perform operations on ciphertext without knowing the key is termed as homomorphic encryption. Till now we have considered the possibility of the server being curious, trying to find out about the data distribution of the participants and the curious participants trying to find out about other participants contributed data. With homomorphic encryption, the possibility of the participants using the model for their own purposes is reduced as they dont have the encryption key. The server sends the homomorphically encrypted model to the data sources. The computations performed in this case are training on the data set present at the data source. There are different levels of homomorphic encryption available. We have used the python-paillier module to homomorphically encrypt the model at the server side. Python-paillier is a Python 3 library for Partially Homomorphic Encryption which follows 3 rules - encrypted numbers can be multiplied by a non encrypted scalar, encrypted numbers can be added together and encrypted numbers can be added to non-encrypted scalars. Partially Homomorphic Encryption is better than Fully Homomorphic Encryption scheme because they are homomorphic with respect to only one kind of operation, either addition or multiplication. The downside with using this is that it converts the model weights into homomorphically encrypted objects which cannot be fed into the keras neural networks and require a customised neural network implementation.

4.3 Applied Algorithms

4.3.1 Federated Averaging

We have used the Federated Averaging algorithm to combine the local Adam training results from each financial institution client with each communication round where the central server performs model averaging. The federated averaging algorithm suggests taking the average of the weight updates received from all the client financial institutions after every communication round. The averaged weight update is then used as the replacement of the original weight vector of the model at the central server. In the consequent round this weight vector is again sent to all client financial institutions.Thus after the required number of communication rounds, the model at the central server is ready to be used for credit risk assessment.

Federated Averaging Algorithm

Select a C- fraction of clients on each round, and compute the gradient of the loss over all the data held by these clients. In each iteration, the server sends a model to C-fraction of clients. Each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

4.3.2 Adam

For performing training on each client financial institution we have used Adam optimization algorithm instead of the standard Stochastic Gradient Descent algorithm. A single learning rate (alpha) is used in Stochastic Gradient Descent for all weight updates and the learning rate does not change during training. In Adam a learning rate is maintained for each network weight (parameter) .This learning rate is separately adapted as the learning unfolds. Thus individual adaptive learning rates are computed by the method for different parameters from estimates of first and second moments of the gradients. Adam as combines the advantages of two other extensions of stochastic gradient descent. Specifically:

- Adaptive Gradient Algorithm (AdaGrad) : It maintains a per-parameter learning rate which improves performance on problems with sparse gradients.
- Root Mean Square Propagation (RMSProp) : It also maintains per-parameter learning rates which are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm performs good on online and non-stationary problems (e.g. noisy).

Adam comprises of the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam

also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates [16].

4.3.3 Stochastic Gradient Descent (SGD)

The word stochastic means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called âĂIJbatchâĂİ which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

We have used Stochastic Gradient Descent in our system.

4.4 Framework Implementation

The framework has been written in Python 3 using python socket io for the server client communications which allows for an event based communication between client server. Python socket io can work both synchronously as well as asynchronously. Both the entities can emit

events to the other, passing data, if required. On emitting an event, the other entity can provide a callback function for the event to be invoked as a way of acknowledgement. A threshold specifying the minimum number of participants present for the round is given. Ideally, all the participants i.e affiliated financial institutions have to participate in each round. Usually in federated learning, a fixed number of devices are randomly selected but the total number of devices present are in high numbers. In our case, we assume the number of participants to be limited, for e.g: In India, there are approximately 100 different banks. With that estimate, all the participating banks or financial institutions can easily participate in all rounds of training. The server waits for the connections from all the participants (threshold is equal to the max number of participants). Once the connections have been established, the server sends the model structure as well as the weights to the clients. The clients on receiving the model and the weights, convert it from json to keras models, set the initial weights of the model as those received from the server and train the model on local data. On completion of training, it sends the training status to the server. The server waits till all the clients send a positive status before proceeding further. There are 2 threads spawned at the server - one for communication with the clients and the other for the task execution. While the server is waiting for the clients to complete the training process, the task execution thread sleeps while the communication thread is still running in the background.

Further, the server calls an event on the clients to send their generated diffie hellman keys to the server which the server stores and sends to the other clients to create shared keys. After this, the encrypted su,v created by the clients are sent to the server, which the server sends to the other clients. Finally the clients append their updates to the sum of perturbations and send it to the server, where such perturbed updates from all clients are added, these averaged updates become the new weights of the model. The user interface at the clientside is running on the Flask server which is listening at a different port than the python socketio. For transferring the status messages either pipes or caches or writing in files could be used. Here we have used the pymemcache module.

4.5 System Requirements

4.5.1 Software Requirements

- Operating Systems: MacOS, Windows 7 or above, Linux
- Tools: Atom, JetBrains PyCharm Community Edition
- Language: Python 3
- Packages:
  - flask
  - python-engineio
  - eventlet
  - python-socketio
  - numpy
  - pyDHE
  - pandas
  - tensorflow
  - keras
  - sympy
  - pyaes
  - crypto
  - flask_cors
  - pymemcache

4.5.2 Hardware Requirements

- Processor: An Intel minimum i5, 2400MHz recommended
- RAM: 4 GB
- HDD: 128 GB

4.6 Implementation Code

4.6.1 Secure Aggregation

```
import numpy as np
from numpy import array
import hashlib
import pyaes
import pyDHE
import pandas as pd

class SecureAggregation:
    R = 1000
    shared_keys={}
    pub_keys = {}
    puvs = {}
    suv = []
    updates  = []
    Alice = None

    def __init__(self):
        print("Secure-aggregation-object-made")

    def get_shared_key_length(self):
        return len(self.shared_keys)

    def generate_shared_key(self):
        for key, value in self.pub_keys.items():
            self.shared_keys[key] = self.Alice.update(value)
        return True

    def receive_pub_keys(self, pub_keys, Alice):
```

```
        self.pub_keys = pub_keys
        self.Alice = Alice
        self.generate_shared_key()

    def encryption(self, updates):
        self.updates = updates
        for item in self.updates:
            self.suv.append(np.random.randint(self.R, size=
                item.shape))
        print("suv")
        print(self.suv)
        encrypted_suvs = {}
        print("SHARED␣KEYS␣SIZE:")
        print(len(self.shared_keys))
        for key, value in self.shared_keys.items():
            value = str(value).encode('utf-8')
            key_maker = hashlib.md5(value)
            key_32 = key_maker.hexdigest().encode('utf-8')
            aes = pyaes.AESModeOfOperationCTR(key_32)

            cipher_text = []
            for item in self.suv:
                plain_text = repr(item)
                cipher_text.append(aes.encrypt(plain_text))

            encrypted_suvs[key] = cipher_text
            print(len(self.shared_keys))
        return encrypted_suvs

    def decryption(self,encrypted_suv):
        decrypted_suv = {}
        for key, value in encrypted_suv.items():
```

```
            shared_key = str(self.shared_keys[key])
            key_maker = hashlib.md5(shared_key.encode('utf-8')
                )
            key_32 = key_maker.hexdigest().encode('utf-8')
            aes = pyaes.AESModeOfOperationCTR(key_32)

            decrypted_suv_list=[]
            for item in value:
                decrypted = aes.decrypt(item)
                decrypted_suv_str = decrypted.decode(encoding=
                    'utf-8', errors='replace')
                decrypted_suv_list.append(eval(
                    decrypted_suv_str))
            decrypted_suv[key] = decrypted_suv_list

        for key, value in decrypted_suv.items():
            puv_list = []
            for i in range(0, len(value)):
                puv_list.append(np.subtract(self.suv[i],value[
                    i]))
            self.puvs[key] = puv_list

    def deleteVal(self):
        self.shared_keys = {}
        self.diffie_parameters = {}
        self.pub_keys = {}
        self.puvs = {}
        self.suv = []
        self.updates = []
        self.dimension = []

    def create_update(self):
```

```
        sum_puv = []
        for key, value in self.puvs.items():
            for item in value:
                sum_puv.append(np.zeros_like(item))
            break

        for key, value in self.puvs.items():
            for i in range(0, len(value)):
                sum_puv[i] = np.add(sum_puv[i], value[i])

        print(len(self.updates))
        print(len(self.puvs))
        for i in range(0, len(self.updates)):
            self.updates[i] = np.add(self.updates[i],sum_puv[i
               ])

        updates = pd.Series(self.updates).to_json(orient='
           values')
        return updates
```

4.6.2 Machine Learning Model

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import numpy as np
import pandas as pd
import json
import tensorflow as tf

from privacy.analysis.rdp_accountant import compute_rdp
from privacy.analysis.rdp_accountant import get_privacy_spent
from privacy.dp_query.gaussian_query import
```

```
    GaussianAverageQuery
from privacy.optimizers.dp_optimizer import
    DPGradientDescentOptimizer

tf.flags.DEFINE_float('noise_multiplier', 1.1, 'Ratio␣of␣the␣
    standard␣deviation␣to␣the␣clipping␣norm')
tf.flags.DEFINE_float('l2_norm_clip', 1.0, 'Clipping␣norm')
tf.flags.DEFINE_integer('batch_size', 500, 'Batch␣size')
tf.flags.DEFINE_integer('epochs', 20, 'Number␣of␣epochs')
tf.flags.DEFINE_boolean('dpsgd', True, 'If␣True,␣train␣with␣DP
    -SGD.␣If␣False,␣''train␣with␣vanilla␣SGD.')
tf.flags.DEFINE_float('learning_rate', 0.15, 'Learning␣rate␣
    for␣training')
tf.flags.DEFINE_integer('microbatches', 1, 'Number␣of␣
    microbatches␣' '(must␣evenly␣divide␣batch_size)')

FLAGS = tf.flags.FLAGS

EPOCHS = 1000
BATCH_SIZE = 5000
X = pd.read_csv("x10k.csv")
Y = pd.read_csv("y10k.csv")
test_data = pd.read_csv("x1k.csv")
test_labels = pd.read_csv("y1k.csv")

dp_average_query = GaussianAverageQuery(FLAGS.l2_norm_clip,
    FLAGS.l2_norm_clip * FLAGS.noise_multiplier,FLAGS.
    microbatches)
optimizer = DPGradientDescentOptimizer(dp_average_query,FLAGS.
    microbatches,learning_rate = FLAGS.learning_rate,
    unroll_microbatches = True)
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True,
```

```
    reduction=tf.losses.Reduction.NONE)

model = Sequential()
model.add(Dense(20, input_dim=43, activation="relu",
    kernel_initializer="uniform"))
model.add(Dense(10, activation="relu", kernel_initializer="
    uniform"))
model.add(Dense(1, activation="sigmoid", kernel_initializer="
    uniform"))

model.compile(loss=loss, optimizer=optimizer, metrics=['
    accuracy'])
print(X.shape)
print(Y.shape)
model.fit(X, Y, epochs=EPOCHS, validation_data=(test_data,
    test_labels), batch_size=BATCH_SIZE)
```

4.6.3 Federation Process

Client

```
import sympy
import random
import numpy as np
from numpy import array
import pandas as pd
import json
import pickle

DATA_FILE_X = "../data/xtrain.csv"
DATA_FILE_Y = "../data/ytrain.csv"
EPOCHS = 5
BATCH_SIZE = 5000
```

```
res_file = "../result/clhistory"

global iter

class FLClient:

        def __init__(self,iter):
                print("␣fl_client␣object␣made␣")
                self.model = None
                self.current_model_version = 0
                self.iter = iter
                self.X = pd.read_csv(DATA_FILE_X)
                self.Y = pd.read_csv(DATA_FILE_Y)
                self.updates  = []

        def weights_from_json(self,model_weights_json):
                json_load = json.loads(model_weights_json)
                model_weights_list = np.array(json_load)
                model_weights = []
                for i in model_weights_list:
                        model_weights.append(np.array(i,dtype=
                            np.float32))
                return model_weights

        def set_model(self, model):
                print( "␣setting␣model␣" )
                self.model = model

        def set_weights(self, model_weights):
                print("␣updating␣model␣weights␣")
                self.model.set_weights(model_weights)
```

```
        def train_model(self,model_weights):
                print("␣start␣training␣")
                self.model.compile(loss='binary_crossentropy',
                    optimizer='adam', metrics=['accuracy'])
                history  = self.model.fit(self.X, self.Y,
                   epochs=EPOCHS, batch_size=BATCH_SIZE)
                res_file_handler = open(res_file + str(self.
                   iter),"wb")
                print("SAVED␣AT␣ITER␣"+str(self.iter))
                pickle.dump(history, res_file_handler)
                res_file_handler.close()
                self.updates = self.get_updates(model_weights)
                print("␣end␣training␣")
                return self.updates

        def get_updates(self, model_weights):
                print(self.model.get_weights())
                updates = [(i-j) for (i,j) in zip(self.model.
                   get_weights(),model_weights)]
                return updates
```

Server

```
import sympy
import numpy as np
import json
import pyDHE

class FLServer:
        encrypted_suvs_clientwise = {}
        def __init__(self):
                print("fl_server␣object␣made")
```

```
def pass_model_parameters(self):
        parameters = "parameters"
        return parameters

def deleteVal(self):
        self.encrypted_suvs_clientwise = {}

def weights_from_json(self, model_weights_json):
        json_load = json.loads(model_weights_json)
        model_weights_list = np.array(json_load)
        model_weights = []
        for i in model_weights_list:
                model_weights.append(np.array(i,dtype=
                  np.float32))
        return model_weights

def averaging(self, updates, count_clients):
        for key, value in updates.items():
                updates[key] = self.weights_from_json(
                  value)
        sum_updates = []
        for key, value in updates.items():
                for item in value:
                        sum_updates.append(np.
                          zeros_like(item))
                break

        for key, value in updates.items():
                for i in range(0, len(value)):
                        sum_updates[i] = np.add(
                          sum_updates[i], value[i])
```

```
            for i in range(0, len(sum_updates)):
                    sum_updates[i] = sum_updates[i] /
                        count_clients

            print("PRINTING SUM UPDATES:")
            print(sum_updates)
            self.deleteVal()
            return sum_updates

    def perturb_util1(self, dict):
            for key, value in dict.items():
                    self.encrypted_suvs_clientwise[key] =
                        {}
            for key, value in dict.items():
                    for key_in, value_in in value.items():
                            if(key != key_in):
                                    self.
                                        encrypted_suvs_clientwise
                                        [key_in][key] =
                                        value_in
            return self.encrypted_suvs_clientwise
```

4.6.4 Connectivity

Client

```
import socketio
import json
import pyDHE
import time
from pymemcache.client import base
from secure_aggregation import SecureAggregation
from flclient import FLClient
```

```
from keras import backend as K
from keras.models import model_from_json
import tensorflow as tf

sio = socketio.Client()

fl_client = None
sa_client = SecureAggregation()
mem_client = base.Client(('localhost', 11211))
mem_client.set('connection',"Connection␣not␣established")
mem_client.set('model_parameters',"Waiting␣for␣others␣to␣
    connect")
mem_client.set('training_status',"Training␣Received␣Model␣on␣
    Local␣Data")
mem_client.set('updates_status',"Federated␣Averaging␣in␣
    Process")
mem_client.set('clear_round_success','Server␣averaging␣the␣
    updates')
mem_client.set('username',"")
mem_client.set('password',"")
mem_client.set('model','')

var = 0
Alice = None
pkey = None
credentials = {}
updates = []
iter = 0


@sio.on('connect')
def on_connect():
```

```
        global Alice, pkey, Alice_server, pkey_server,
            credentials

        Alice = pyDHE.new()
        pkey = Alice.getPublicKey()
        while not mem_client.get('username') or not mem_client
            .get('password'):
                time.sleep(5)
        print('␣connection␣established␣')
        credentials['username'] = mem_client.get('username').
            decode('utf-8')
        credentials['password'] = mem_client.get('password').
            decode('utf-8')
        sio.emit('authenticate', credentials)
        mem_client.set('connection',"Connection␣Successfully␣
            Established")


@sio.on('receive_averaged_model')
def receive_averaged_model(model_string):
        mem_client.set('model',model_string)
        sio.emit('disconnect')

@sio.on('message')
def on_message(data):
        print('␣data␣received␣at␣client␣' + data)
        sio.emit('message','This␣message␣has␣been␣successfully
            ␣received')
        #fl_client.process_message(data)

@sio.on('disconnect')
def on_disconnect():
```

```
        global credentials
        credentials['username'] = None
        credentials['password'] = None
        mem_client.set('connection',"Connection not
            established")
        mem_client.set('model_parameters',"Waiting for model
            to be received")
        mem_client.set('training_status',"Training Received
            Model on Local Data")
        mem_client.set('updates_status',"Federated Averaging
            in Process")
        mem_client.set('clear_round_success','Server averaging
            the updates')
        print(' disconnected from server ')

@sio.on('send_perturbs')
def on_send_perturbs(data):
        global updates
        print(data)
        while not updates:
                time.sleep(5)
        suv_dict = sa_client.encryption(updates)
        sio.emit('receive_perturb',suv_dict)

@sio.on('wait_shared_key')
def on_wait_shared_key(data):

        skey = sa_client.get_shared_key_length()
        while(skey < data):
                time.sleep(5)
                skey = sa_client.get_shared_key_length()
```

```
@sio.on('receive_pub_keys')
def receive_pub_keys(pub_keys):
        global Alice
        print("Received public keys")
        tf = False
        tf = sa_client.receive_pub_keys(pub_keys,Alice)
        #while not tf:
        #       time.sleep(5)
        #tf = False
        sio.emit('shared_key_status','shared keys made')

@sio.on('get_public_keys')
def on_get_public_keys(data):
        global pkey
        sio.emit('receive_public_key',pkey)

@sio.on('clear_round')
def on_clear_round(data):
        global updates
        updates = []
        sa_client.deleteVal()

        mem_client.set('model_parameters',"Waiting for model
           to be received")
        mem_client.set('training_status',"Training Received
           Model on Local Data")
        mem_client.set('updates_status',"Federated Averaging
           in Process")

        mem_client.set('clear_round_success','Round completed'
           )
```

```
@sio.on('receive_model')
def on_receive_model(model_json):
        global updates, count, iter
        print('message␣received␣with␣', model_json)
        mem_client.set('model_parameters',"Model␣downloaded␣
           successfully")
        iter += 1
        fl_client = FLClient(iter)
        #count = fl_client.return_count()
        dict = json.loads(model_json)
        model = model_from_json(json.dumps(dict["structure"]))
        fl_client.set_model(model)
        model_weights = fl_client.weights_from_json(json.dumps
           (dict["weights"]))
        fl_client.set_weights(model_weights)
        updates = fl_client.train_model(model_weights)
        mem_client.set('training_status',"Training␣completed␣
           successfully")
        sio.emit('training_status','training␣done')
        #updates_json = fl_client.get_updates_json(
           model_weights)

        print('␣sending␣updates␣to␣server␣')
        K.clear_session()
        #sio.emit('send_model_updates', updates_json)
        #sio.emit('message','This message has been
           successfully received')

@sio.on('receive_suvs')
def on_receive_suvs(encrypted_suv_clientwise):
        sa_client.decryption(encrypted_suv_clientwise)
        sio.emit('get_updates', sa_client.create_update())
```

```
            mem_client.set('updates_status',"Updates␣sent␣back␣to␣
                server␣successfully")

sio.connect('http://192.168.43.248:8004')
```

Server

```
import engineio
import eventlet
eventlet.monkey_patch()
import socketio
import time
import json
import numpy as np
import pyDHE
import retry
from flserver import FLServer
import pandas as pd
from keras.models import model_from_json


res_file = "../result/serverres"

NO_OF_ROUNDS = 20


sio = socketio.Server(async_mode='eventlet')
app = socketio.Middleware(sio)

Bob = pyDHE.new(18)
pkey = Bob.getPublicKey()


count_clients = 0
count_rounds = 0
count_auth_clients = 0
```

```
conn_threshold  = 3
update_threshold = 3
updates_received = 0
client_updates = {}
server_wait_time = 5
suv_dictionary = {}
count_train_done = 0
count_shared_done = 0
fin_weights_str = retry.model_weights_json
fin_struct = retry.model_json

fin_weights = []
credentials=[{'username':'$un@in@','password':'passit'},{'
   username':'priya','password':'priy@'}]
shared_keys = {}
fl_server = FLServer()

pub_keys = {}

@sio.on('connect')
def connect(sid, environ):
    global count_clients
    count_clients+=1
    print('connect')
    client_updates[sid] = ""

@sio.on('authenticate')
def authenticate(sid, dict):
    global credentials, count_auth_clients
    found = True
    print('connection authenticated')
    count_auth_clients+=1
```

```
@sio.on('message')
def message(sid, data):
    print('message ', data)

@sio.on('disconnect')
def disconnect(sid):
    global count_clients
    count_clients -=1
    client_updates.pop(sid)
    print('disconnect', sid)

@sio.on('get_updates')
def get_updates(sid, update):
    global updates_received, client_updates
    updates_received += 1
    client_updates[sid] = update

@sio.on('receive_public_key')
def receive_public_key(sid, data):
    global pub_keys
    pub_keys[sid] = data

@sio.on('receive_perturb')
def receive_perturb(sid,suv_dict):
    global suv_dictionary
    suv_dictionary[sid] = suv_dict

@sio.on('training_status')
def training_status(sid, data):
    global count_train_done
    count_train_done+=1
```

```
    print("train status: " + data)
    #secure_agg()

@sio.on('shared_key_status')
def shared_key_status(sid, data):
    global count_shared_done
    count_shared_done+=1
    print("shared status: "+ data)

def connServ():
    eventlet.wsgi.server(eventlet.listen(('', 8004)), app)

def send_model():
    global count_clients, conn_threshold, count_train_done,
       fin_weights, fin_weights_str, count_auth_clients
    if count_clients >= conn_threshold and count_clients ==
       count_auth_clients:
        print('threshold reached')
        #model_parameters = fl_server.pass_model_parameters()
        sio.emit('receive_model', '{"structure" : ' +
           fin_struct + ',"weights" : ' + fin_weights_str + '}
           ')
        fin_weights = fl_server.weights_from_json(
           fin_weights_str)
        while count_train_done < count_clients:
            eventlet.greenthread.sleep(seconds=5)
        return True
    return False

def diffie_hellman():
    global pub_keys, count_shared_done
    sio.emit('get_public_keys','send me public keys')
```

```
    while len(pub_keys)<count_clients:
        eventlet.greenthread.sleep(seconds=5)
    sio.emit('receive_pub_keys',pub_keys)
    while count_shared_done < count_clients:
        eventlet.greenthread.sleep(seconds=5)


def secure_agg():
    global count_clients, suv_dictionary
    print("Inside␣secure␣aggregation")
    diffie_hellman()
    sio.emit('send_perturbs','Send␣me␣the␣perturbations')
    while len(suv_dictionary) < count_clients:
        eventlet.greenthread.sleep(seconds=5)
    encrypted_suv_clientwise = fl_server.perturb_util1(
       suv_dictionary)
    for key, values in encrypted_suv_clientwise.items():
        sio.emit('receive_suvs',values,room=key)


def federating_process():
    global count_clients, updates_received, update_threshold,
       client_updates, suv_dictionary, fin_weights,
       fin_weights_str, count_rounds
    while count_rounds<NO_OF_ROUNDS:
        eventlet.greenthread.sleep(seconds=5)
        if send_model():
            secure_agg()
            while updates_received < count_clients:
                eventlet.greenthread.sleep(seconds=5)

            sum_updates = fl_server.averaging(client_updates,
               count_clients)
```

```
            for i in range(0,len(sum_updates)):
                fin_weights[i] = np.add(fin_weights[i],
                    sum_updates[i])
            fin_weights_str = pd.Series(fin_weights).to_json(
                orient='values')
            sio.emit('clear_round','Clear␣the␣round␣rn')
            for key, value in client_updates.items():
                client_updates[key] = ""
            updates_received = 0
            suv_dictionary = {}
            count_train_done = 0
            count_shared_done = 0
            count_rounds += 1
            print(count_rounds,"
                ----------------------------------------ROUND
                -COMPLETED
                ----------------------------------------
                ")

    print("ALL␣ROUNDS␣DONE")
    sio.emit('receive_averaged_model','{"structure":' +
        fin_struct + ',␣"weights":' + fin_weights_str + '}')

if __name__ == '__main__':
    pool = eventlet.GreenPool()
    pool.spawn(connServ)
    pool.spawn(federating_process)
    pool.waitall()
```

4.6.5 Homomorphic Encryption

Client

```
import phe as paillier
import json

class HEClient:
    def __init__():
        print("HE Object made")

    def convert_to_json(self,enc_data):
        self.enc_with_one_pub_key['pubkey'] = {'n': pubkey.n}
        enc_with_one_pub_key['values'] = [[(str(x.ciphertext()
            ),x.exponent) for x in y] for y in data]
        serialised = json.dumps(enc_with_one_pub_key)
        return serialised

    def convert_from_json(self,serialised):
        received_dict = json.loads(serialised)
        pk = received_dict['pubkey']
        public_key_rec = paillier.PaillierPublicKey(n=int(pk['
            n']))
        enc_nums_rec = [[ paillier.EncryptedNumber(
            public_key_rec, int(x[0]), int(x[1])) for x in i]
            for i in received_dict['values']]
        return enc_nums_rec
```

Server

```
import phe as paillier
import json

class HEServer:
    def __init__(self):
        self.pubkey, self.privkey = paillier.
            generate_paillier_keypair(n_length=1024)
```

```
        self.enc_with_one_pub_key = {}

    def encryption(self,data):
        enc_data = [[self.pubkey.encrypt(i) for i in x] for x
            in data]
        return enc_data

    def convert_to_json(self,enc_data):
        self.enc_with_one_pub_key['pubkey'] = {'n': pubkey.n}
        enc_with_one_pub_key['values'] = [[(str(x.ciphertext()
            ),x.exponent) for x in y] for y in data]
        serialised = json.dumps(enc_with_one_pub_key)
        return serialised

    def convert_from_json(self,serialised):
        received_dict = json.loads(serialised)
        pk = received_dict['pubkey']
        public_key_rec = paillier.PaillierPublicKey(n=int(pk['
            n']))
        enc_nums_rec = [[ paillier.EncryptedNumber(
            public_key_rec, int(x[0]), int(x[1])) for x in i]
            for i in received_dict['values']]
        return enc_nums_rec

    def decryption(self,enc_nums_rec):
        data = [[privkey.decrypt(i) for i in x] for x in
            enc_nums_rec]
        return data
```

4.6.6 Output

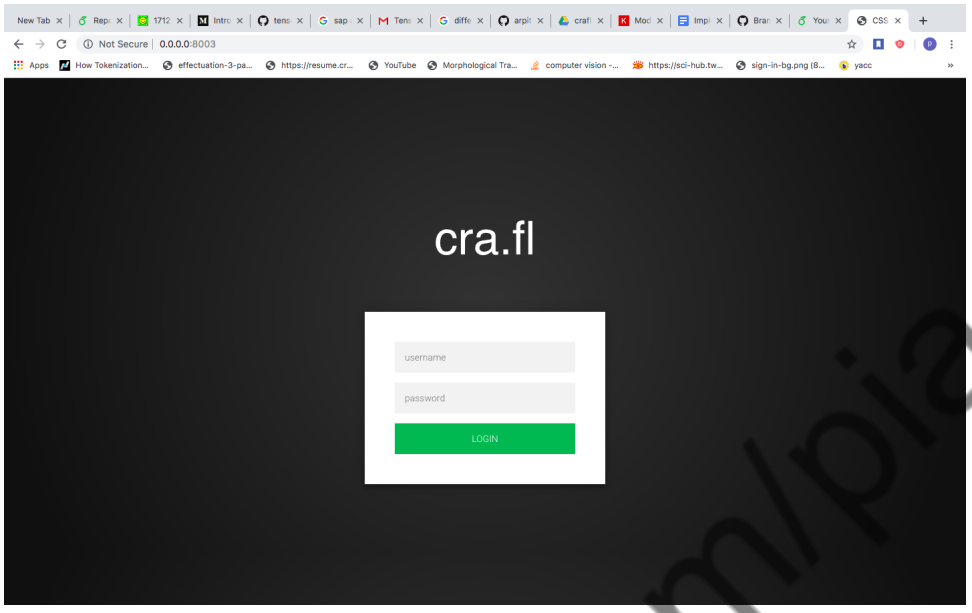


Figure 4.1: Authentication Page on Client

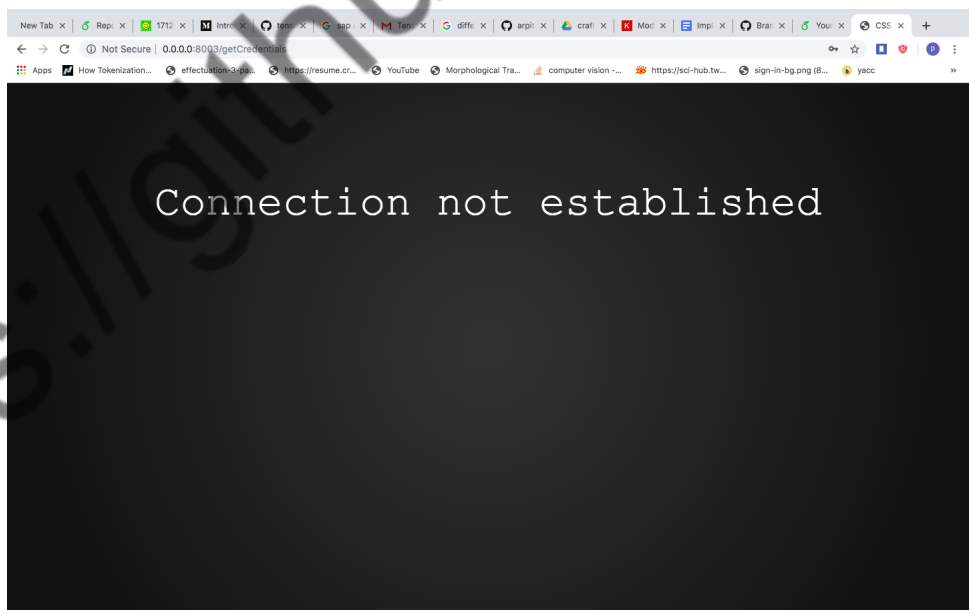


Figure 4.2: Wrong Credentials on Client

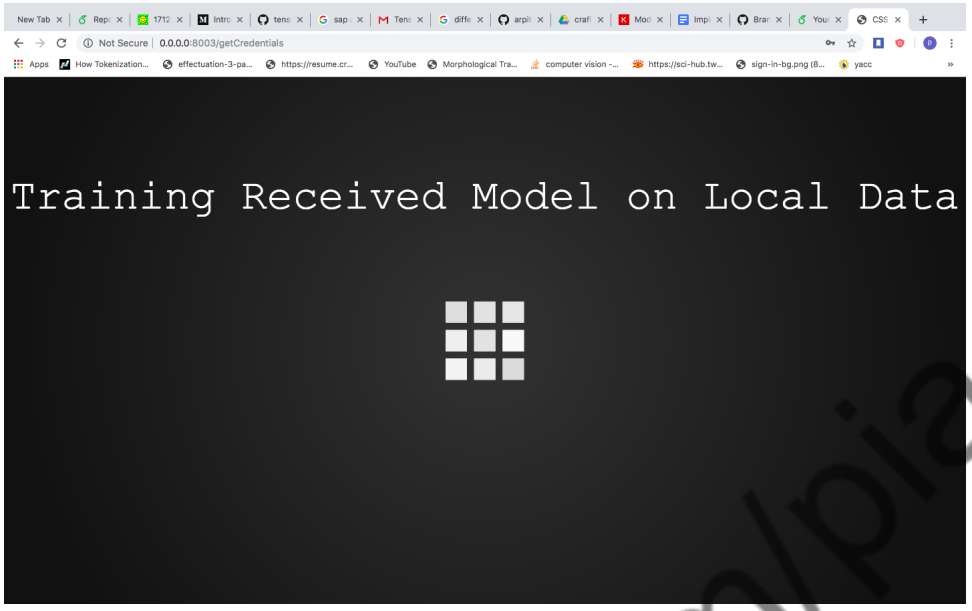


Figure 4.3: Model Received from Server, Training on Client

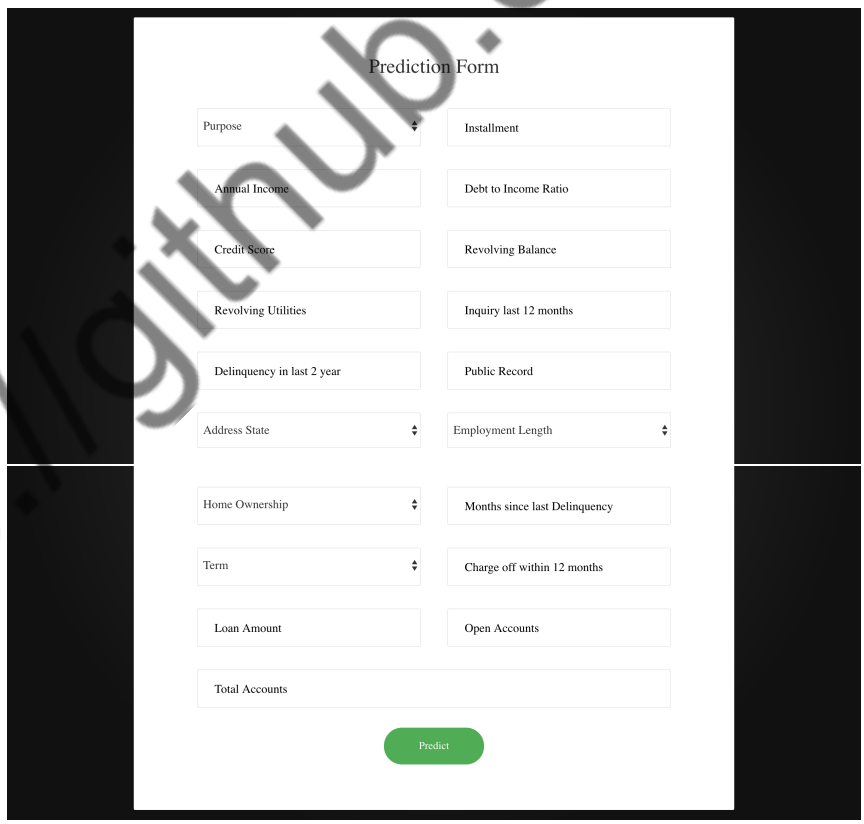


Figure 4.4: Prediction on Trained Model

CHAPTER 5

RESULT

5.1 Without Differential Privacy

The training accuracy vs epochs for different iterations was plotted and studied for each client. We found that, overall, the accuracy increased with epochs.

5.1.1 Clients

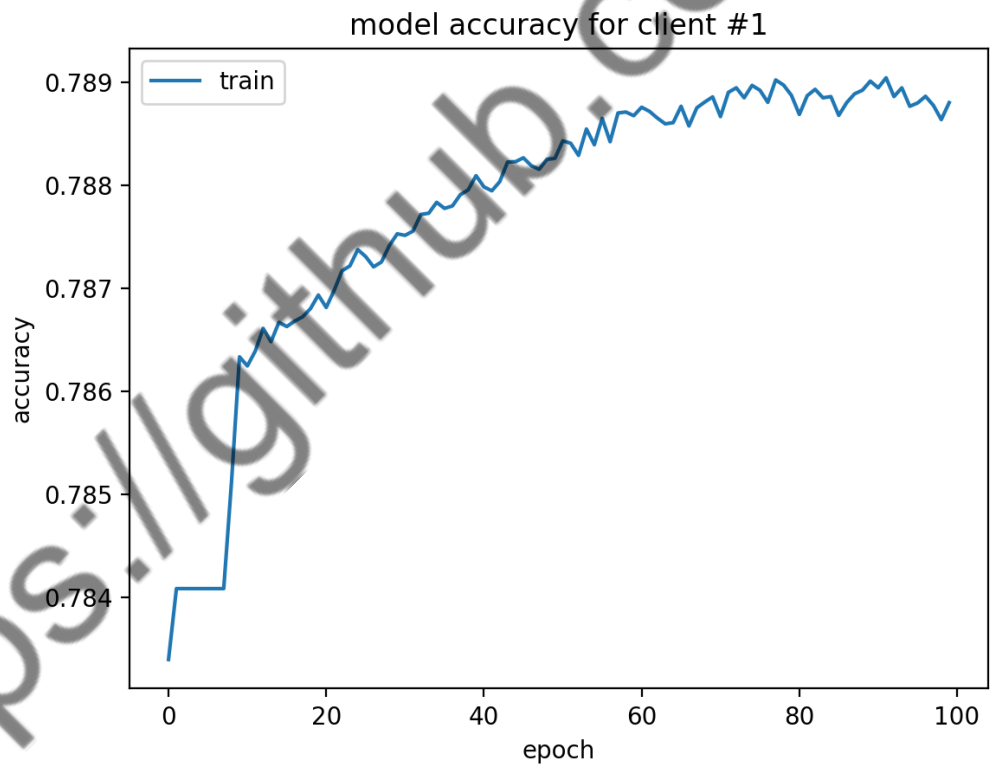


Figure 5.1: Client #1 without Differential Privacy

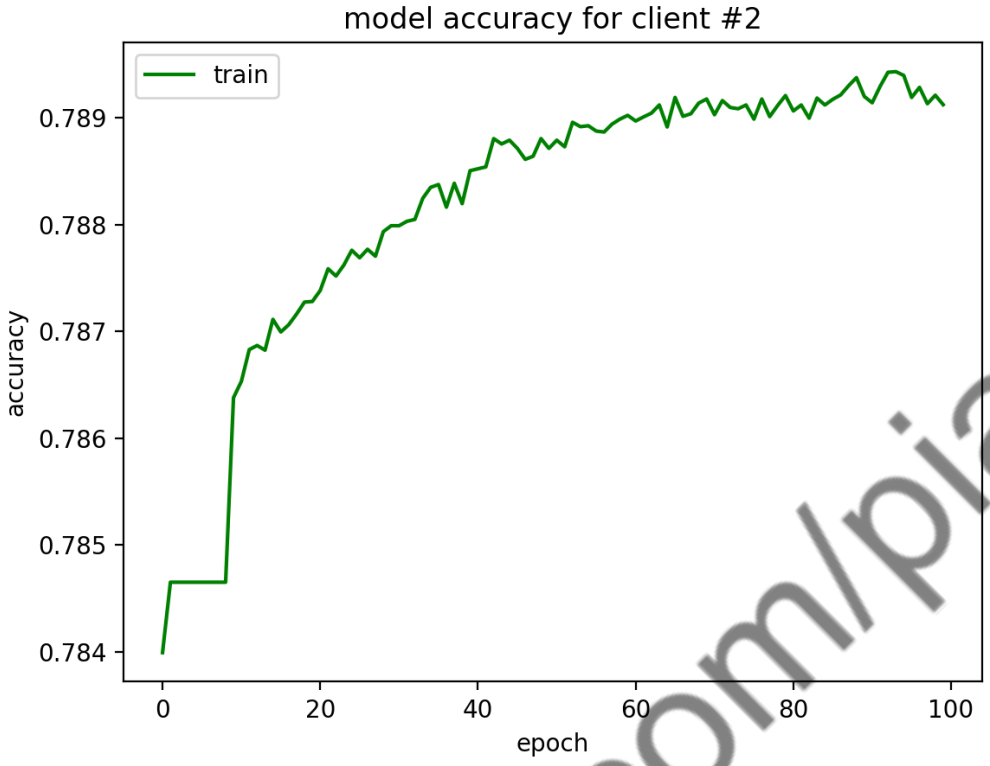


Figure 5.2: Client #2 without Differential Privacy

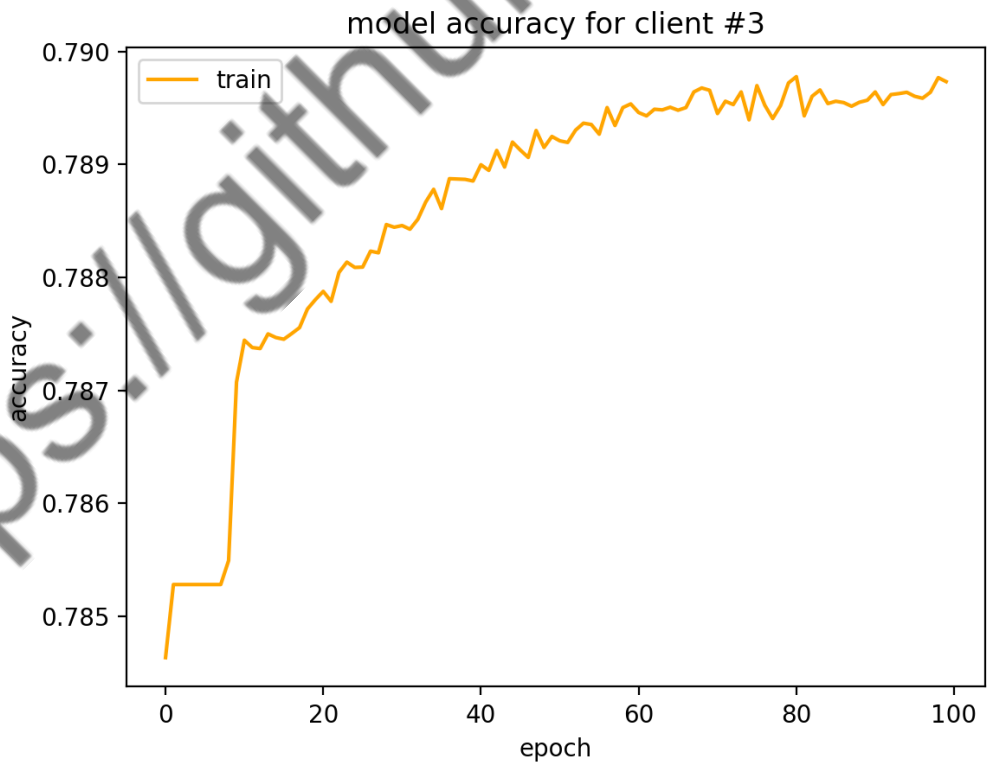


Figure 5.3: Client #3 without Differential Privacy

5.2 With Differential Privacy

The training accuracy vs epochs for different iterations was plotted and studied for each client. We found that, overall, the accuracy increased with epochs. With Differential Privacy, we found that, the accuracy fluctuates a lot with epochs but eventually results in higher accuracy.

5.2.1 Clients

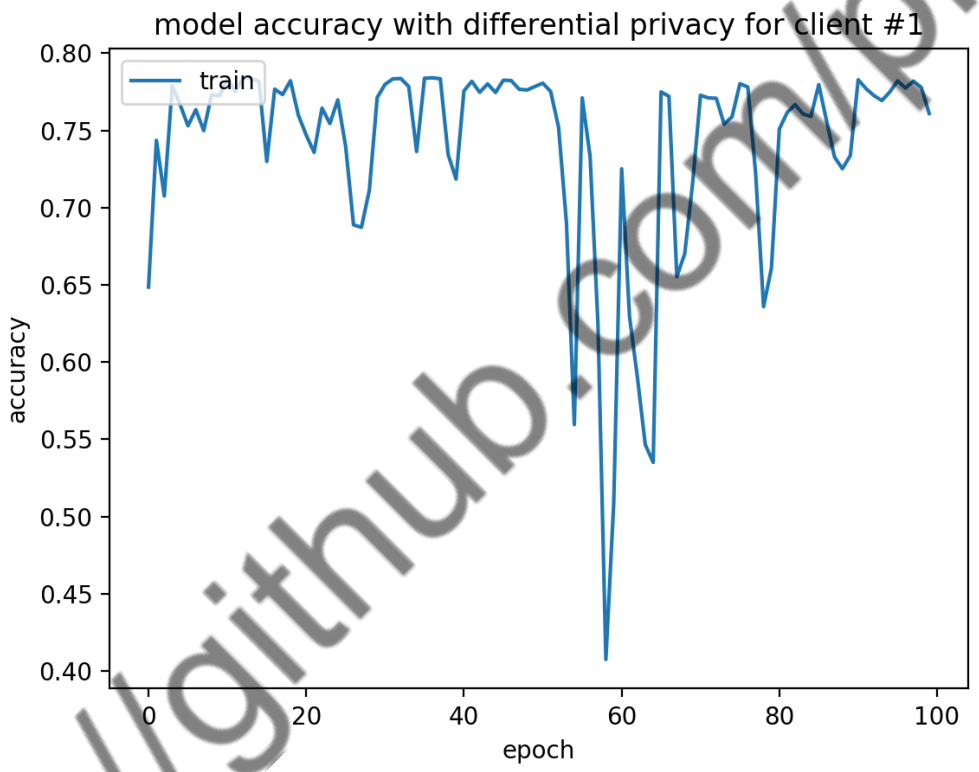


Figure 5.4: Client #1 with Differential Privacy

Figure 5.5: Client #2 with Differential Privacy

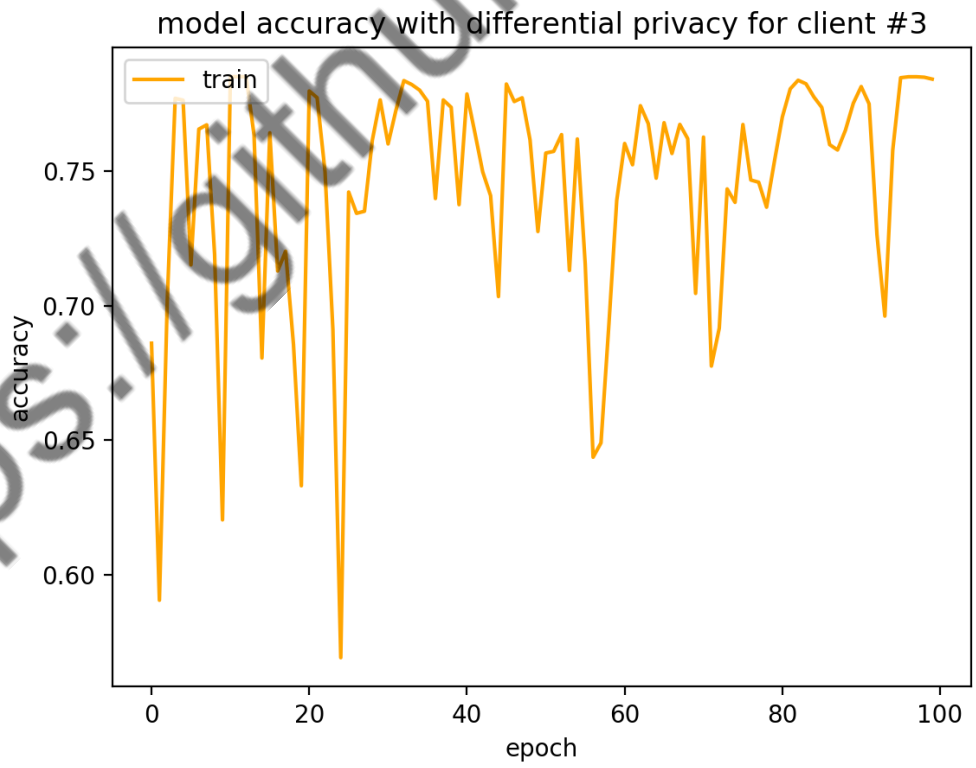


Figure 5.6: Client #3 with Differential Privacy

Table 5.1: Accuracies

Accuracy List	
Training Method	Accuracy
Independent ML	72.4
Centralised ML	79.34
Federated Learning w/o Differential Privacy	78.52
Federated Learning with Differential Privacy	78.29

5.2.2 Server

At each iteration, we calculated the accuracy of the model with testing data. We found that, although the accuracy fluctuates a lot at different iterations, it eventually increases.

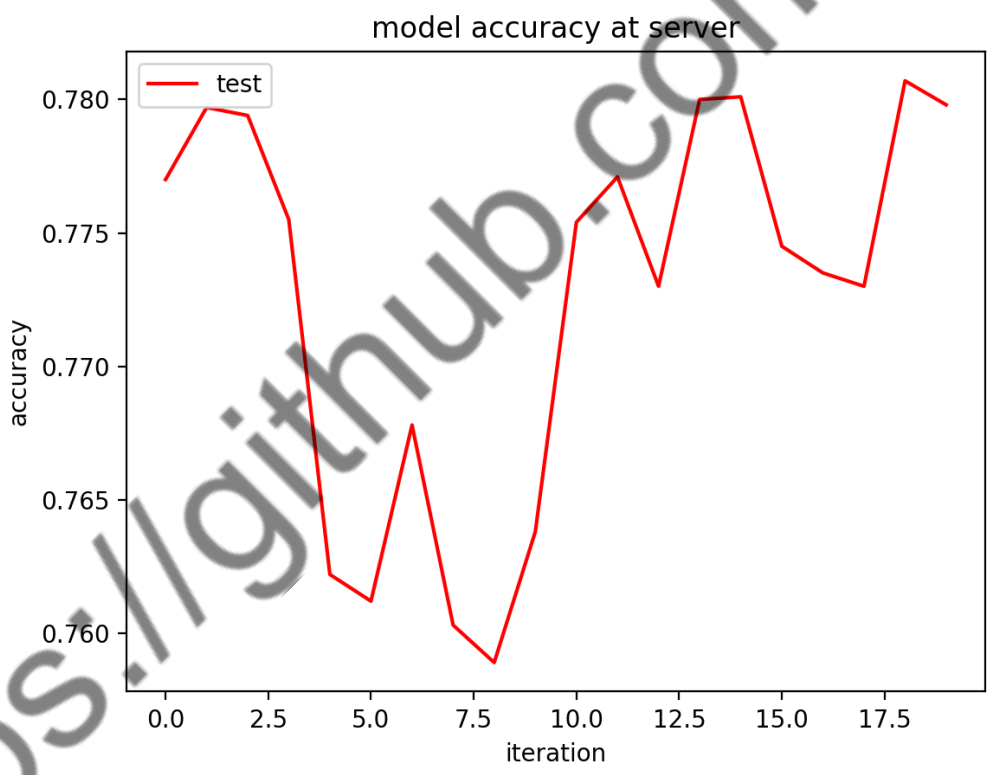


Figure 5.7: Testing the Model at the server made by federation

5.3 Comparison of Accuracies

CHAPTER 6

CONCLUSION

In this work, we have analyzed and presented the current practices of credit risk assessment used by banks. We have proposed the use of federated learning for learning a credit risk assessment model on data from a large number of banks. We discovered that the method is better than independent training on self data by financial institutions. The accuracy was better for federated learning compared to independent training. Federated Learning does not perform better than training on centrally collected data, but this method has its own limitations considering the privacy of data and communication cost. The consequent model on the central server is more accurate, owing to the higher amount of data used in training. The proposed solution has the following advantages :

- There is no need to use complicated technology for collection, storage and ensuring the security of data as there is no collection of data at a central location.
- The data is credible and trusted as it comes from verified participating banks.
- Since the customer data is sensitive, banks are usually reluctant to share data with other parties. In this approach, the data doesnâĂŹt leave the bank and hence privacy is ensured.

The system is also secured by using secure aggregation, differential privacy and homomorphic encryption mechanisms. This model can be utilized by all the participating client banks creating a win-win situation of being able to avail the benefits of a better model without having to compromise the privacy of the data.

REFERENCES

[1] Credit risk assessment with a multistage neural network ensemble learning approach, https://www.sciencedirect.com/science/article/pii/S0957417407000206?via%3Dihub

[2] Credit Risk, https://en.wikipedia.org/wiki/Credit_risk

[3] Credit Analysis, https://en.wikipedia.org/wiki/Credit_analysis#cite_ref-2

[4] Credit Score Analysis, https://opensiuc.lib.siu.edu/cgi/viewcontent.cgi?article=2074&context=gs_rp

[5] Credit Bureau, https://www.investopedia.com/terms/c/creditbureau.asp

[6] Ping Wang, Christopher Johnson (2018) âĂIJCybersecurity Incident Handling: A Case of Equifax Data BreachâĂİ , http://www.iacis.org/iis/2018/3_iis_2018_150-159.pdf

[7] Federated Learning: Collaborative mMachine Learning without Centralized Training Data, http://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[8] Naman Agrawal, Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, Brendan McMahan (2017). âĂIJCommunication Efficient Learning of Deep Networks from Decentralized DataâĂİ, https://arxiv.org/pdf/1602.05629.pdf

[9] Jakub Konecny, H. Brendan McMahan, Felix X. Yu, Ananda Theertha Suresh, Dave Bacon (2017) âĂIJFederated Learning: Strategies for Improving Communication EfficiencyâĂİ , https://arxiv.org/abs/1610.05492

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, Karn Seth (2016) âĂIJPractical Secure

Aggregation for Federated Learning on User - Held DataâĂİ , https://arxiv.org/pdf/1611.04482.pdf

[11] Sudhamathy G. âĂIJCredit Risk Analysis and Prediction Modelling of Bank Loans Using RâĂİ , http://www.enggjournals.com/ijet/docs/IJET16-08-05-414.pdf

[12] Ahmad Ghodselahi(2011) âĂIJHybrid Support Vector Machine Ensemble Model for Credit ScoringâĂİ , http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.4051&rep=rep1&type=pdf

[13] Slawomir Goryczka, Li Xiong, Vaidy Sunderam(2011) âĂIJSecure Multiparty Aggregation with Differential PrivacyâĂİ , http://www.mathcs.emory.edu/predict/pub/secure_pais13.pdf

[14] Daisy P.K(2016) âĂIJA Study on Credit Information Bureau (India) LimitedâĂİ , https://eprawisdom.com/jpanel/upload/articles/1221pm19.Daisy%20P.K.pdf

[15] Robin Geyer, Tassilo Klein and Moin Nabi (ML Research Berlin), Differentially Private Federated Learning: A Client Level Perspective, https://medium.com/sap-machine-learning-research/client-sided-differential-privacy-preserving-federated-learning-1fab5

[16] Diederik P. Kingma, Jimmy Lei Ba, ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, https://arxiv.org/pdf/1412.6980.pdf