Wafaa Kanakri, Baudouin M. Ramazani

1 Introduction

Short Message Service (SMS) is the text communication service component of phone, web, or mobile communication systems. According to the International Telecommunication Union (ITU) the usage of the communication based on SMS has become a massive commercial industry [4]. Many users use SMS as a main tool to complete financial transactions, send passwords and confidential information. This makes SMS as a main target to Spam messages that can be utilized by hackers to make frauds and many attacks. Artificial Intelligence field is one of the promising fields to solve this problem. But standard machine learning approaches require to save the training data on one machine which acts as a data center. These approaches have a common weakness point due to the leak in saving the privacy and security of raw data communication. Strict rules are applied in all countries around the world to save users' personal privacy and data security like the General Data Protection Regulation (GDPR) enforced by the European Union on May 25, 2018 [3].

Secure federated machine learning is a promising solution to these challenges. Federated machine learning is a distributed learning paradigm that aims to train machine learning models from scattered and isolated data [1]. Using this technique users' devices will be used as computing nodes performing computation on their local data then their computation results will be used to update a global model [2]. Based on this secure efficient and intelligent approach, machine-learning models can be built collaboratively across many different devices while saving their privacy and preventing data leakage. Therefore, Federated machine learning principal is to train through model aggregations rather than data aggregation and keep local data stay at the local device. Utilizing federated machine learning model will be very effective and useful to build a system that predicts and filter SMS spam messages and at the same time saves users' privacy.

Our model will contain three main components: The data set, the learning coordinator, and the contributor clients (mobile phone users). The learning coordinator will create a training task to classify the SMS messages as spam or ham. The provider will produce and embed an initial global model into the SMS application. Each round, the initial global model will select randomly a subset of the mobile phones that will perform the training process locally. For each training round each client, takes one step of gradient descent using its local SMS data. After each round, the model update is submitted by each participating mobile phone to the server. The server collects the updates from the client mobile phones and executes model aggregation that will update the global model. The updated model will be broadcasted to the mobile phone devices for the next training round. Fig.1 below illustrates the infrastructure of federated learning scheme.

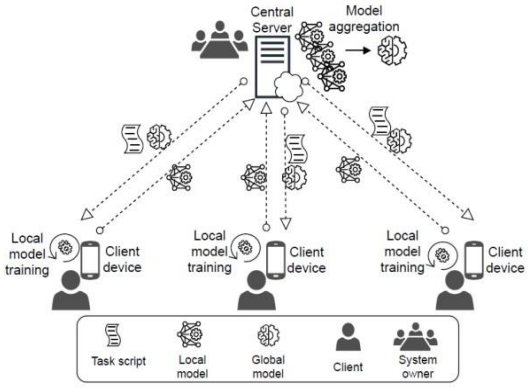

Fig.1. Federated learning schematic illustration of the infrastructure [5]

2 Related work

There are growing interest by researchers in SMS spam filtering. We are targeting two topics, first is the SMS classification algorithm that will be used to classify the messages as spam or ham. Second is the federated machine learning that will benefit in terms of scalability and privacy. Some of the characteristics of the SMS messages introduce challenges to build robust antispam filters. For example, informal language used in writing SMS messages; short length which causes a difficulty in classification and filtration [6]. The authors in [6] applied different machine learning algorithms to real SMS spams from UCI Machine Learning repository. New features were added to classify the messages such as the length of messages in terms of number of characters beside saving the old features like words occurrence. Based on the simulation results, the author found that multinomial naive Bayes with Laplace smoothing and SVM with linear kernel are among the best classifiers for SMS spam detection. But this work had not been carried out on public SMS spam datasets, which is important to get real, general, robust, and valid classification result. Moreover, it does not figure out the privacy and the centralized storage challenges of SMS messages classification and spam detection. In [4] the authors used clustermedoid spam filtering (CMSF) technique to classify a group of patterns into related classes. The authors used this technique which is unsupervised learning tool on e-mail spam datasets that had true labels. Two types of clustering methods that have been used for spam classification are density-based clustering and K-nearest neighbors (kNN). They showed that this technique performed better than semi-supervised techniques like dynamic markov compression (DMC). On the other hand, [8] proposed a lightweight deep neural model called Lightweight Gated Recurrent Unit (LGRU) for SMS spam detection. they proposed the Matthews correlation coefficient (MCC) as the evaluation measure instead of the total cost ratio (TCR). They showed that MCC provides more balanced evaluation of the prediction than total cost ratio measure since this measure is more useful when there is class imbalance. Although they get better performance, they did not solve the scalability issue and saving the privacy of the users. The authors in [4] introduced a real public non-encoded SMS spam collection to get reliable dataset. A collection of 425 SMS spam messages was manually extracted from a forum in which cell phone users make public claims about SMS spam messages. The authors provided tokens frequencies and finding the most relevant words in terms of information gain scores. Also, they applied different established machine learning methods to classify the messages as spam or ham. They found that Support Vector Machine outperforms other evaluated classifiers. However, their extracted dataset cannot be considered as public since it is from limited sources. At the same time due to privacy and limited computational resources it is difficult to get one public centralized dataset.

To solve the data privacy challenge that need to be used in machine learning algorithms challenges, some studies propose to use federated machine learning which is considered as privacy preserving, decentralized collaborative machine learning [1]. In [1] the authors introduced a federated learning framework which includes horizontal federated learning, vertical federated learning, and federated transfer learning. They provide architectures for the federated-learning framework. In [11] applied federated machine learning on real-world domain adaptation dataset for email spam detection. They classified whether an email in a user's inbox is spam or ham and personalized the spam filtering for each user. They simulated 15 users collaborating to build a spam classifier in a supervised setting. Their results show that federated learning provides the best overall accuracy, comparing to other models. This work had been applied on email dataset not SMS messages, but it showed promising results. In [11] demonstration of federated learning in a resource constrained networked environment is shown. A federated learning system deployed in an emulated wide area communication. The resources availability is dynamic, heterogeneous, and intermittent. Authors use a server that holds the global network model trained, and multiple clients holding different local data, which are called local models

3 Methodology

3.1 Data set

The unprocessed data set contains SMS messages as plain text. We transform the raw text data into a set of features that describe qualitative concepts in a quantitative way. In our project, that will be a binary strategy coding. Zero represents spam or one represents ham. Historically, to extract features from a text, the most important type of feature that's been used is word count [23]. The source of the dataset in our project is UCI Machine Learning Repository. The SMS Spam Collection consists of 5,574 English SMS messages that were labeled as spam or ham. The text file of the dataset contains one message per line. Each line is composed by two columns; v1 contains the label which is ham or spam and v2 contains the raw text. This dataset has been collected for SMS Spam research. 425 SMSs spam messages were extracted manually from the UK forum called Grumbletext [12], where the mobile phone users have the facility to claim publicly about SMS spam messages. The source of a random subset of 3,375 SMS is NUS SMS Corpus (NSC) [13], which is a dataset of about 10,000 legitimate messages that were collected for research purposes at the Department of Computer Science at the National University of Singapore. Most of the messages originated from volunteer students attending the University in Singapore who were aware that their contributions will be made publicly available. A subset of 450 SMS ham messages collected from Caroline Tag's PhD Thesis which is available at [14]. Lastly, the SMS Spam Corpus v.0.1 Big incorporated which has 1,002 SMS ham messages and 322 spam messages. This Corpus is public available at[15].

Data preprocessing is done to eliminate the incomplete, noisy and inconsistent data [16]. This stage will help in maximizing classifier performance. Preprocessing data involves three steps. First step, all the characters will be represented in lowercase so the word which may be written in different cases will be treated as the same word. For example, 'prize' and 'PRIZE' must be treated as one word. Second step, tokenizing each message in the dataset by chopping it into pieces and removing the punctuation characters. A token is an instance of a sequence of characters in some specific document that are formed together as a useful semantic unit for processing [17]. Last step will be removing the stop words. Stop words are frequently occurring words that hardly carry any information and orientation [16].
Two encodings which are commonly used in spam filtering problems will be used to represent messages in each category: Bag-of-Words (BoW) and the Term Inverse Frequency of Document Frequency (TFIDF) [17]. BoW and TFIDF are simplified representations that map words within a document to a vector of frequencies. The vector is indexed by a vocabulary. The benefit of these mappings that they capture lexical features while ignoring syntax or semantics [21]. The Tf-idf approach works on the assumption that importance of a word is inversely proportional to how often it occurs across all the documents [20]

3.2 Algorithm

This project uses Naïve Bayes classifier as the local machine learning algorithm. The main classification principle for this algorithm is based on searching through the text for words that are either more likely to occur in spam or ham messages. When a word is noticeably more likely to occur in one context rather than the other, its occurrence can be considered as evidence to classify a new message as spam or ham. A Naïve Bayes classifier is based on the application of the Bayes' theorem. Bayes' law gives the probability of the occurrence of an event given some conditions or other events. The adjective naive comes from the assumption that the features in a dataset are mutually independent [20]. The dictionary is the set of every word collected from every single message in our dataset. There are two major steps to consider that will considerably reduce computation power. The first is to rework and simplify the math behind it. The classifier will have a slightly different form after simplification, but the

principle remains the same. Calculating the probability means checking for the occurrence frequency of each word in the message being classified to belong to the spam message. The classifier takes spam and every word from the message as events. Therefore, the Naïve Bayes' theorem described above becomes:

$$P(spam|w_1, w_2, w_3, ..., w_n) = P(spam)\frac{P(w_1|spam) * P(w_2|spam) * P(w_3|spam) * ... * P(w_n|spam)}{P(w_1) * P(w_2) * P(w_3) * ... * P(w_n)} \quad (1)$$

$$P(spam|w_1, w_2, w_3, ..., w_n) = P(spam)\frac{\prod_{i=1}^{n} P(w_i|spam)}{\prod_{i=1}^{n} P(w_i)} \quad (2)$$

The denominator being a constant in every iteration, we can simply remove it because it only scales every probability with the same factor but does not affect how they compare to each other at all. Moreover, to successfully word test the words in the testing dataset, which are nonexistent in the training dataset, we will use additive smoothing [20]. By applying additive smoothing we add a number alpha to the numerator and the denominator as shown in equation 3. We are considering the Laplace smoothing, which implies alpha equals 1. [20].

Therefore, the final expressions we are considering will be:

P(spam): the part of spam messages in our dataset

P(w_i|spam): The probability of a word to be found in the spam messages

$$P(w_1|spam) = \frac{N_{w_i|spam} + \alpha}{N_{spam} + \alpha * N_{dictionary}} \quad (3)$$

$N_{dictionary}$: Total unique words in the dataset.

N_{spam}: Total number of words in the spam messages.

$N_{w_i|spam}$: Number of words repeated in all spam messages.

Next step is to train the spam classifier model on remote edge devices without having any access to their private data. We will use PySyft to implement the decentralized machine learning training model. PySyft is an open-source framework that enables secured, private computations in deep learning, by combining federated learning and differential privacy in a single programming model integrated into different deep learning frameworks such as PyTorch, Keras or TensorFlow [21]. PySyft decouples private data from model training, using Multi-Party Computation (MPC) within PyTorch. For simulation purposes we are going to split the original data into 70% as training dataset and 30% as testing dataset. Then each of these datasets will be separated equally into two halves and will be sent to two remote machines to simulate remote and private data. The remote machines will interact with the application using the global machine learning model. PySyft framework we can simulate these remote machines by using the abstraction of a Virtual Worker object. Then a simple Gated Recurrent Unit (GRU) created with sigmoid activation for classification task. In the training step we will use FederatedDataLoaders that iterates over the batches of data that come from several devices, in a federated manner. Then, for each epoch we are going to calculate the training loss and the validation loss. In addition to that, we will calculate the Area Under the Region Of Convergence (ROC) Curve score since the dataset is unbalanced, where 87% of the messages are ham while only 13% are labeled as spam. After each round, the model update is submitted by each participating mobile phone to the server. The server

collects the updates from the client mobile phones and executes model aggregation that will update the global model. The updated model will be broadcasted to the mobile phone devices for the next training round. This process will be repeated until the global model converges.

3.2 Training and Testing

The original data set split into training and testing datasets to validate the classification model. The train dataset will be used during the learning process to generate a good predictive model, while the test dataset will be used to test the precision of the model. The length of each message will be added as a new feature beside the words' occurrence feature. The learning coordinator will create a training task to classify the SMS messages as spam or ham. The provider will produce and embed an initial global model into the SMS application. Each round, the initial global model will select randomly a subset of the mobile phones that will perform the training process locally. For each training round each client, takes one step of gradient descent using its local SMS data. After each round, the model update is submitted by each participating mobile phone to the server. The server collects the updates from the client mobile phones and executes model aggregation that will update the global model. The updated model will be broadcasted to the mobile phone devices for the next training round. This process will be repeated until the global model converges.

4 Implementation

The implementation is divided in two stages. The first stage is implementing Naïve Bayes classifier as the local machine learning algorithm. The second stage is implementing federated learning model. Federated learning enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on their local devices.

4.1 Naïve Bayes classifier implementation

We implemented the Naïve Bayes classifier as the local machine learning algorithm, taking strong assumption that the messages are independent, and the words included in the SMS messages are also independent. This model is oblivious to the context of the message. Based on ignoring the correlations among inputs, we used the bag of words model to represent a text document as a feature vector. The main three criteria for feature extraction in our model are salient, invariant, and discriminatory. Salient relates to being aware that the extracted features are important and meaningful with respect to the problem domain. Invariance concerns with that the features must be insusceptible to distortion, scaling, and orientation. While discriminatory related to satisfy that the selected features bear enough information to distinguish well between patterns when used to train the classifier. By apply these three criteria we achieve dimensionality reduction and good accuracy. The steps in this stage are as follow:

Data splitting

The dataset is a labeled message text classified as spam or ham with 75:25 proportion. Both data sets are randomly selected. To get a representative sample of the population and to avoid skewed results, the original class proportions are preserved in both the training and testing datasets. This makes the training, and the testing sets better reflect the properties of the original dataset.

Data pre-processing

Data preprocessing consists of removing the punctuations and the stop-words from the feature space. Punctuations presumed to be uninformative in representing the content of a text. The Natural Language Toolkit (NLTK), a popular Python library for NLP, provides common stop words. The words converted to lower case since the same word with different letter cases has the same meaning, and we need to treat them as one term in the vocabulary. The second step is tokenizing the corpus. Tokenization is usually

defined as the process of splitting the document up into multiple meaningful pieces or tokens. Regular expression is used in our implementation to extract the tokens from the corpus. The extracted tokens will be used to construct the vocabulary set in the classification model and extract features to train the model.

Feature engineering and extraction method

The first step in the feature is creating the set of vocabulary. The vocabulary is a collection of all unique tokens that occur in the training data set, where each word is associated with the count of how it occurs. The order of the words in the document doesn't matter. Then, the vocabulary set has been used to construct the Term Document Frequency Matrix which consists of the feature vectors. This is a mapping of every word to its message and the number of times the word occurs in the message. Applying this concept, we can convert a collection of documents to a matrix, with each document being a row and each token being the column, and the corresponding (row, column) values being the frequency of occurrence of each word or token in that document. Building a dictionary in python representing the same concept of the Term Document Matrix. Every term in the vocabulary represents a key, and for each key there is a vector that identifying the messages in which that term appears in, and the frequency of their appearance. Our data set is a large collection of 5,572 rows of text data. There are 9688 unique words represent the vocabulary.

Training and obtaining prediction results

Given a set of messages $M = \{m_1, m_2, \ldots, m_j, \ldots, m_{|M|}\}$ and category set $C = \{spam\ (c_s), ham(c_h)\}$, where m_i is the i^{th} message in M and C is the possible label set. The main task in the training and prediction stage is building a Boolean categorization function $\Phi_{spam}(m_j, c_i): M \times C \rightarrow \{True, False\}$. If $\Phi(m_j, c_i)$ is True, this means that message m_j belongs to category c_i; otherwise, m_j does not belong to c_i. Using the precalculated term document matrix, the classifier $\Phi_{spam}(m_j)$ is applied to the vector representation of a message m to predict whether m is spam or not. In our implementation we calculated the prior probabilities; $P(spam)$, $P(ham)$. Independently, they were estimated as the frequency of occurrences of documents belonging to the categories $\{spam, ham\}$ in the training dataset M. Based on Bayes' theorem
and the theorem of the total probability, the probability that a message with vector $w_1, \ldots, w_n$ belongs to a category $c_i \in \{c_s, c_h\}$ is:

$$P(c_i|m) = P(c_i)P(m|c_i)/P(m) \quad (4)$$

Naive Bayes filter classifies each message in the category that maximizes $P(c_i) * P(m|c_i)$.
Taking the representation of the message as a set of words and based on the "Naive" assumption, the words in a message are conditionally independent and the order they appear is irrelevant. then,

$$P(c_i|w_1, ..., w_n) = P(c_i) * P(w_1|c_i) * P(w_2|c_i) * P(w_n|c_i) \quad (5)$$

After that, we built two dictionaries: one for spam and the other for ham. Using these two dictionaries we mapped each word to its probability of appearing in the respective label. We computed the parameters $P(w_i|spam)$ $P(w_i|ham)$ where w_i stands for each word in the vocabulary. The result is the "likelihood" which is the product of the individual probabilities of seeing each word in the set of spam or ham messages. In the case a word has never been met during the learning phase, both the numerator and the denominator are equal to zero. To handle this case, we apply Laplace additive smoothing technique that helps tackle the problem of zero. We added a number alpha to the numerator and add alpha times number of classes over which the probability is found in the denominator. Finally, after computing the probability of such message in the testing dataset being spam and ham, the classification decision will be based on the following formal decision rule.

$$\hat{y} = arg \max_{(k \in Spam, Ham)} \; p(C_k)) \prod_{i=1}^{n} p(w_i|C_k) \qquad (6)$$

This rule considers the larger value as the classification result for the message

Evaluating the model

To evaluate the performance of our model a confusion matrix is used. We calculated the precision, the recall and f1 score metrics. F1-Score is the weighted average of Precision and Recall. Our model manages to get a 99% accuracy on the training dataset and an optimal precision, recall, and F1-Score with values.

4.2 Federated Learning using PySyft

The federated learning algorithm implemented by building SMS spam filtering model that achieved data integration which are compliant with data privacy and security laws, beside improving the model performance. The main idea is to save the training datasets distributed across multiple devices (workers) to prevent data leakage while the model is shared between the workers.

Initiating virtual workers with PySyft

We are using federated learning with the PyTorch extension of PySyft for a classification task with a simple 1-layer GRU. Firstly, we are simulating two remote machines by using the abstraction of Virtual Worker object. Virtual Workers are all live on the same machine and do not communicate over the network. They simply replicate the chain of commands and expose the very same interface as the actual workers to communicate with each other [24]. Each machine has a similar number of labeled SMS messages.

Loading and sending data to the workers

After loading and splitting the training dataset, the .federate method is invoked to transform the training dataset into a Federated Dataset. Then, this federated dataset is sent to the Federated DataLoader. The DataLoader iterates over remote batches. The testing dataset remains unchanged since the local client will perform the test evaluation.

Creating Gated Recurrent Unit (GRU) model with sigmoid activation for classification task

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate [25]. In this step we will set hypermeters and structure of the networks. Then, we use fully connected with Rectifier function (ReLU) as an activation function to down-sampling from the unit layer. According to the Classifying layer, we use fully connected with Sigmoid function as an activation function to predict whether it is a spam or a normal message.

GRU: Gated Hidden State

The main difference between vanilla RNNs and GRUs is that the latter supports hidden state gating. This means that we have separate processes for updating and resetting hidden states. These strategies are learned and handle the above-mentioned concerns. If the first token, for example, is extremely important, we will learn not to change the concealed state after the first observation. Similarly, we will learn to ignore momentary insights that are irrelevant to our goals. Finally, we'll learn how to reactivate the latent state as necessary. Below, we go over this in greater depth. [26]

GRU: Reset Gate, Update Gate and Candidate Hidden State

The reset and update gates are the first things we really need to cover. We make the vectors with elements in the range (0,1) so that we may execute convex combinations with them. A reset gate, for example, would allow us to choose how much of the prior state we want to remember. Similarly, an update gate would let us control how much of the new state is a duplicate of the old. We'll start by designing these gates. Given the input of the current time step and the hidden state of the previous time step, Fig. 2 depicts the inputs for both the reset and update gates in a GRU. Two fully connected layers with a sigmoid activation function provide the outputs of the two gates [26].

GRU: Dropout, Embedding, Fully connected, Sigmoid Layer

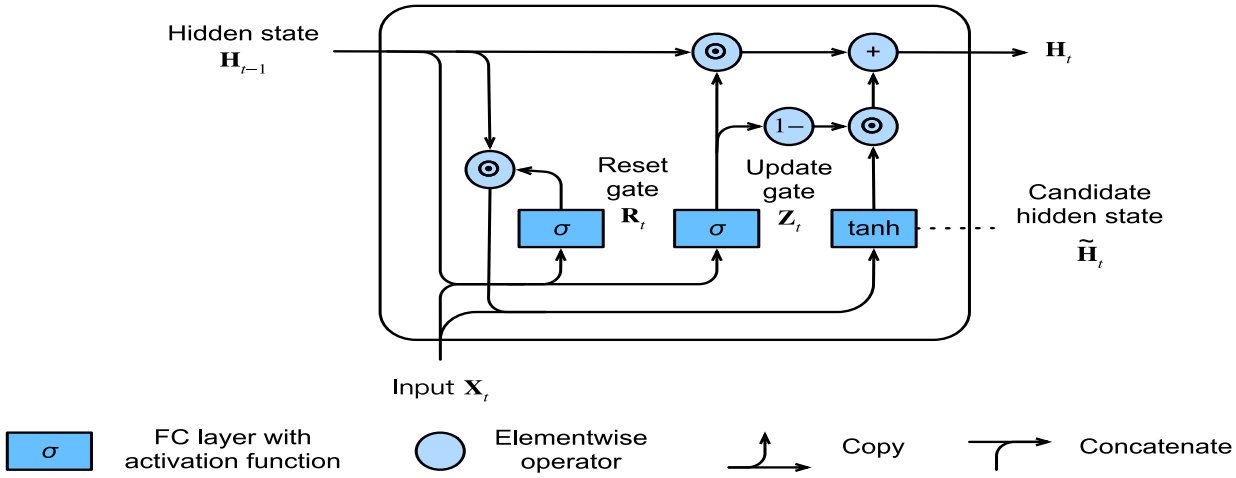

Fig. 2. Computing the hidden state in a GRU model

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h), \tag{7}$$

R_t : Reset gate $\in R^{n \times h}$
Z_t : Update gate $\in R^{n \times h}$
X_t : Minibatch Input $R^{n \times d}$

d: Number of inputs
W: weight $\in R^{h \times h}$
Br,bz: biases $R^{1 \times h}$

Training

Hence spam filtering task consists of a binary classification, the Binary Cross Entropy Loss is used. The training and validations losses is computed for each epoch. In addition to calculating the Area Under the ROC Curve score since the dataset is unbalanced, where 87% of the messages are ham comparing to 13% as spam. Because the dataset is distributed across two machines, we will send the model for each batch for training using model.send() function. For model improvement, getting back the updated model and the loss using .get() method. The server collects the updates from the client mobile phones and executes model aggregation that will update the global model. The updated model will be broadcasted to the mobile phone devices for the next training round. This process will be repeated until the global model converges.

5. Results

1. Naïve Bayes Classifier Results
   A. Misclassification

The model is oblivious to context hidden in the messages. It only considers the word individually and checks if that word can be spam or not. Whereas it is noticed that the words individually do not tell the whole story. The words: call, phone, connected, messages, executive, mobile, etc. are similar to words found usually in spam messages. In short, the model misclassified all those ham messages that had a more formal tone to the spam messages.

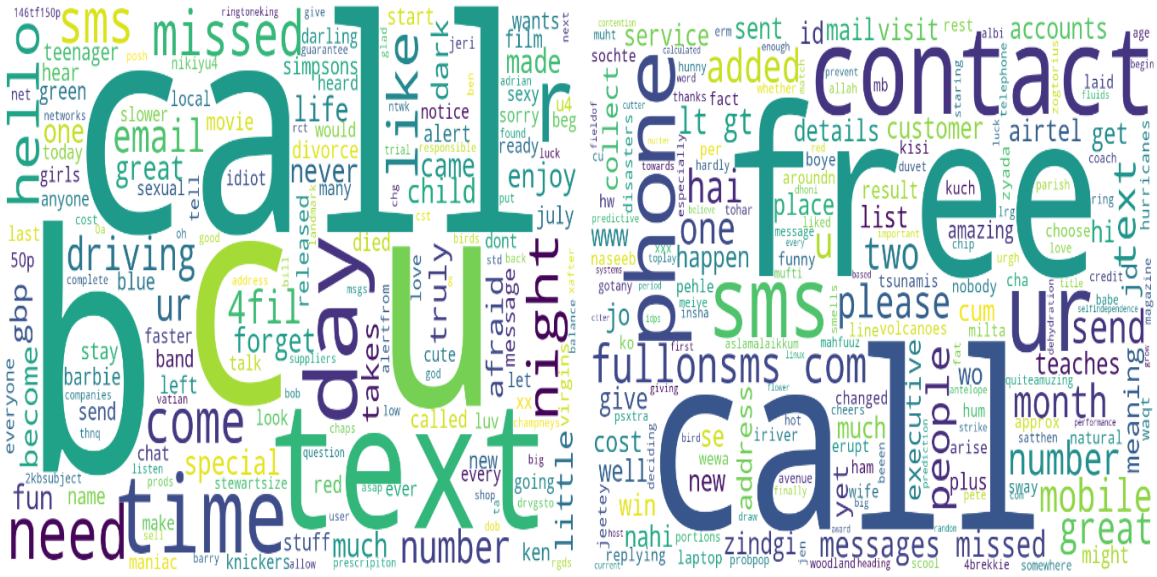

Fig.3. Words Misclassification

   B. Results Validation & Confusion Matrix

A confusion matrix is used to give information about the True positives and the False negatives (correct predictions) and the True negative and False positive (incorrect predictions) of the model. Since accuracies give just a general idea about the model but NOT accurate judgment. The sklearn.metrics module provides the inbuilt function classification_report which is generated based on the true labels and the predicted labels. Precision is the fraction where the messages are ham/spam out of all the messages the model classified to be ham/spam.

	Precision	Recall	F1-score	Support
Ham	0.99	1.00	0.99	4782
Spam	0.98	0.92	0.95	792
Accuracy			0.99	5574
Macro Average	0.98	0.96	0.97	5574
Weighted Average	0.99	0.99	0.99	5574

Table 1. Confusion Matrix

2. Federated Learning: Binary Classification
   A. Binary classification

Binary classification refers to classification issues having two class labels. The HAM case is represented by the False class(Class of 0's), while the SPAM case is represented by the True class(Class of 1's) in our binary classification system.

B. Variables Values

1. Batch Size: The quality of the model is reduced when a bigger batch is used, as evaluated by its capacity to generalize. Small-batch approaches, on the other hand, reliably converge. A very small batch size also results in inaccurate models; a large batch size, on the other hand, will not fit in memory and will an incredible amount of time. This is why a batch size of 32 was chosen.
2. Embedding/Hidden Dimension: The availability of computational resources is one of the most important elements in determining the appropriate embedding dimension. This explains why 50 and 10 were chosen, respectively.
3. Clip: Gradient Clipping was set to 5 to deal with explosion problem which is frequent in recurrent neural networks.
4. Learning rate: The value of 0.1 for the learning rate was kept.

Variable	Value
Epochs	25
CLIP	5
Learning rate	0.1
Batch Size	32
Embedding Dimension	50
Hidden Dimension	10
Dropout	0.2

Table 2. Model Configurations

C. Epochs Results:

As it is shown, the model learns after every epoch until it reaches number twelve. On top of that, the results can be validated by looking at the way both the training and validation losses linearly vary over every epoch. On top of that, an area under the curve is computes to confirm how well the system was training. after every epoch.

```
Epoch 1/25...     AUC: 66.251%...     Training loss: 0.44806...     Validation loss: 0.38368
Epoch 2/25...     AUC: 73.781%...     Training loss: 0.37157...     Validation loss: 0.36242
Epoch 3/25...     AUC: 77.963%...     Training loss: 0.35123...     Validation loss: 0.34213
Epoch 4/25...     AUC: 81.678%...     Training loss: 0.32937...     Validation loss: 0.31559
Epoch 5/25...     AUC: 84.462%...     Training loss: 0.29779...     Validation loss: 0.28946
Epoch 6/25...     AUC: 87.564%...     Training loss: 0.27290...     Validation loss: 0.26229
Epoch 7/25...     AUC: 90.562%...     Training loss: 0.23494...     Validation loss: 0.22733
Epoch 8/25...     AUC: 93.527%...     Training loss: 0.20014...     Validation loss: 0.18822
Epoch 9/25...     AUC: 95.279%...     Training loss: 0.16376...     Validation loss: 0.16056
Epoch 10/25...     AUC: 96.406%...     Training loss: 0.13459...     Validation loss: 0.13685
Epoch 11/25...     AUC: 96.985%...     Training loss: 0.11637...     Validation loss: 0.12477
Epoch 12/25...     AUC: 97.109%...     Training loss: 0.10309...     Validation loss: 0.12328
Epoch 13/25...     AUC: 97.316%...     Training loss: 0.09348...     Validation loss: 0.11641
Epoch 14/25...     AUC: 97.247%...     Training loss: 0.08909...     Validation loss: 0.11292
Epoch 15/25...     AUC: 97.373%...     Training loss: 0.07704...     Validation loss: 0.10983
Epoch 16/25...     AUC: 97.485%...     Training loss: 0.07085...     Validation loss: 0.11043
Epoch 17/25...     AUC: 97.286%...     Training loss: 0.06218...     Validation loss: 0.11354
Epoch 18/25...     AUC: 97.307%...     Training loss: 0.05564...     Validation loss: 0.11545
Epoch 19/25...     AUC: 97.204%...     Training loss: 0.05111...     Validation loss: 0.11452
Epoch 20/25...     AUC: 97.228%...     Training loss: 0.04915...     Validation loss: 0.11947
Epoch 21/25...     AUC: 97.361%...     Training loss: 0.04640...     Validation loss: 0.11371
Epoch 22/25...     AUC: 97.249%...     Training loss: 0.04152...     Validation loss: 0.12221
Epoch 23/25...     AUC: 97.057%...     Training loss: 0.03879...     Validation loss: 0.11990
Epoch 24/25...     AUC: 97.094%...     Training loss: 0.03652...     Validation loss: 0.12111
Epoch 25/25...     AUC: 97.008%...     Training loss: 0.03221...     Validation loss: 0.13848
```

Fig. 4. Results

Conclusion

In conclusion, unlike traditional machine learning algorithms that need the training data to be saved on a single computer that serves as data center, the model developed in this study delivers the same outcomes while not having direct access to the local data. Regular procedures have a common flaw in saving the privacy and security of raw data. In the first portion, the research proposes a strong Naïve Bayes Classifier that displays an accuracy of 98% and 99% for Spam messages and Ham messages, respectively. On the other half, the research focuses on the design of a GRU model that is sent to multiple virtual workers holding data that the main computer does not have access to. The model sent is trained by the federated training data and updated according to the label data structure, which holds the validation data. The research led to interesting AUC results of above 97%. The area under the ROC curve (AUC) is

a global measure of the ability of a test to discriminate whether a specific condition is present or not present. An AUC of 0.5 represents a test with no discriminating ability (ie, no better than chance), while an AUC of 1.0 represents a test with perfect discrimination [27]. In our results on figure 4, the system learns and display better accuracy after every epoch until it reaches 97%, our final accuracy of the area under the curve. The results are obtained using the variables on table 2. This defines how well the system was trained given the data, the model, and the configurations of the model.

As a result, the binary classification employed in spam filtering is the Binary Cross Entropy Loss. For each epoch, the training and validation losses are calculated. In addition to computing the Area Under the ROC Curve score, because the dataset is uneven, with 87 percent of the messages classified as spam and just 13% as ham. Because the dataset is split across two computers, model.send() function was used to deliver the model for each batch to be trained. Using the .get() function, the main system receives the updated model and loss for model improvement. The server gathers information from client mobile phones and performs model aggregation, which updates the global model. For the next training cycles, the new model will be disseminated on mobile phone devices. This procedure will be continued until the global model has reached a point of convergence.

References

[1] Yang, Qiang, et al. "Federated machine learning: Concept and applications." ACM Transactions on Intelligent Systems and Technology (TIST) 10.2 (2019): 1-19.

[2] Ge, Suyu, et al. "Fedner: Privacy-preserving medical named entity recognition with federated learning." arXiv preprint arXiv:2003.09288 (2020).

[3] https://gdpr-info.eu

[4] Almeida, Tiago A., José María G. Hidalgo, and Akebo Yamakami. "Contributions to the study of SMS spam filtering: new collection and results." Proceedings of the 11th ACM symposium on Document engineering. 2011.

[5] Lo, Sin Kit, et al. "Architectural patterns for the design of federated learning systems." arXiv preprint arXiv:2101.02373 (2021).

[6] Liu, Xiaoxu, Haoye Lu, and Amiya Nayak. "A Spam Transformer Model for SMS Spam Detection." IEEE Access 9 (2021): 80253-80263.

[7] Whissell, John S., and Charles LA Clarke. "Clustering for semi-supervised spam filtering." Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference.
2011.

[8] Wei, Feng, and Trang Nguyen. "A lightweight deep neural model for SMS spam detection." 2020 International Symposium on Networks, Computers and Communications (ISNCC). IEEE, 2020.

[9] F. Lai, X. Zhu, H. Madhyastha and M. Chowdhury, "Oort: Efficient Federated Learning via Guided Participant Selection", Electronics, vol. 9, no. 9, p. 1359, 2020. Available: 10.3390/electronics9091359

[10] A. Imteaj, I. Khan, J. Khazaei and M. Amini, "Demonstration of Federated Learning in a ResourceConstrained Networked Environment", Electronics, vol. 10, no. 16, p. 1917, 2020. Available:
10.3390/electronics10161917

[11] Peterson, Daniel, Pallika Kanani, and Virendra J. Marathe. "Private federated learning with domain adaptation." arXiv preprint arXiv:1912.06733 (2019).

[12] http://www.grumbletext.co.uk

[13] https://scholarbank.nus.edu.sg

[14] https://etheses.bham.ac.uk/id/eprint/253/1/Tagg09PhD.pdf

[15] http://www.esp.uem.es/jmgomez/smsspamcorpus

[16] https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

[17] Kowsari, Kamran, et al. "Text classification algorithms: A survey." Information 10.4 (2019): 150.

[18] https://kb.iu.edu/d/aepu

[19] https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

[20] Raschka, S. "Naive Bayes and text classification I: introduction and theory. arXiv e-prints." arXiv preprint arXiv:1410.5329 (2014).

[21] Rojas-Galeano, Sergio. "Using BERT Encoding to Tackle the Mad-lib Attack in SMS Spam Detection." arXiv preprint arXiv:2107.06400 (2021).

[22] ZPySyft: A Library for Easy Federated Learning. In: Rehman M.H.., Gaber M.M. (eds) Federated Learning Systems. Studies in Computational Intelligence, vol 965. Springer, Cham.

[23] Machine learning for hackers. D. Conway, and J. White. O'Reilly Media, Sebastopol, CA, (2012)

[24] Ryffel, Theo, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. "A generic framework for privacy preserving deep learning." arXiv preprint arXiv:1811.04017 (2018).

[25] Poomka, Pumrapee, Wattana Pongsena, Nittaya Kerdprasop, and Kittisak Kerdprasop. "SMS spam detection based on long short-term memory and gated recurrent unit." International Journal of Future Computer and Communication 8, no. 1 (2019).

[26] "9.1. Gated Recurrent Units (GRU) — Dive into Deep Learning 0.17.1 documentation", *D2l.ai*, 2020. [Online]. Available: https://d2l.ai/chapter_recurrent-modern/gru.html. [Accessed: 06- Dec- 2021].

[27] Hoo, Zhe Hui et al. "What Is An Roccurve". *Emj.Bmj.Com*, 2017, https://emj.bmj.com/content/emermed/34/6/357.full.pdf.