Privacy-Preserving Classification with Secret Vector Machines

Valentin Hartmann
EPFL
valentin.hartmann@epfl.ch

Konark Modi
Cliqz
konarkm@cliqz.com

Josep M. Pujol
Cliqz
josep@cliqz.com

Robert West
EPFL
robert.west@epfl.ch

ABSTRACT

Today, large amounts of valuable data are distributed among millions of user-held devices, such as personal computers, phones, or Internet-of-things devices. Many companies collect such data with the goal of using it for training machine learning models allowing them to improve their services. However, user-held data is often sensitive, and collecting it is problematic in terms of privacy.

We address this issue by proposing a novel way of training a supervised classifier in a distributed setting akin to the recently proposed federated learning paradigm [26], but under the stricter privacy requirement that the server that trains the model is assumed to be untrusted and potentially malicious; we thus preserve user privacy by design, rather than by trust. In particular, our framework, called secret vector machine (SecVM), provides an algorithm for training linear support vector machines (SVM) in a setting in which data-holding clients communicate with an untrusted server by exchanging messages designed to not reveal any personally identifiable information.

We evaluate our model in two ways. First, in an offline evaluation, we train SecVM to predict user gender from tweets, showing that we can preserve user privacy without sacrificing classification performance. Second, we implement SecVM's distributed framework for the Cliqz web browser and deploy it for predicting user gender in a large-scale online evaluation with thousands of clients, outperforming baselines by a large margin and thus showcasing that SecVM is practicable in production environments.

Overall, this work demonstrates the feasibility of machine learning on data from thousands of users without collecting any personal data. We believe this is an innovative approach that will help reconcile machine learning with data privacy.

1 INTRODUCTION

With the growing number of smartphones, intelligent cars and smart home devices, the amount of highly valuable data that is spread among many devices increases at a rapid pace. Those devices are typically in possession of end users and so is the data produced by and stored on them. Of course companies are interested in making use of this data, e.g., to detect usage patterns, make better-informed business decisions, and ultimately improve their products. As an example, consider the case of a web browser vendor wanting to infer demographics from users' browsing histories in order to automatically change the default behavior for hiding adult-only content from users inferred to be minors, or to show relevant website suggestions to users based on their inferred age groups.

The classical way of building the necessary prediction model would be to collect the users' data on a central server and then run a machine learning algorithm on it. But this comes with severe disadvantages. First, the user has to put the necessary trust in the data-collecting entity. Even in case of trust, the relationship between the two parties is very imbalanced; new regulations such as the European Union's General Data Protection Regulation (GDPR) [12] and e-privacy frameworks try to rebalance the relationship to be more fair. But still, even in the case of perfect regulation, the collection of user data incurs a privacy risk. There are many ways in which privacy could be compromised: hacks leading to a data breach [40], disgruntled or unethical employees that use the data for their own benefit [10], companies going bankrupt and selling the data as assets [34], and of course government-issued subpoenas and backdoors [17, 30]. All in all, it is safe to assume that gathering users' data puts their privacy at risk, regardless of the comprehensiveness of the data management policies in place.

It is thus desirable to be able to build prediction models without learning any personally identifying information about the individual users whose data is used in the training process. For instance, the party who is training the model should not be able to infer labels or feature values of individual users. This requirement immediately precludes us from using the standard machine learning setup, where feature vectors for all users are stored in a feature matrix, labels in a label vector, and an optimization algorithm is then used to find the model parameters that minimize a given loss function.

The issue with the vanilla machine learning setup is that the party training the model sees all data—features and labels—at the same time, which typically makes it easy to infer user identities, even if the data is pseudo-anonymized, i.e., if actual user ids have been replaced with random identifiers. One way to achieve this is by tying together multiple features associated with the same user, as was the case with the now-infamous AOL data breach, where users thought to be anonymized were identified by analyzing all their search queries together [6]. Moreover, user privacy can also be compromised by correlating the pseudo-anonymized data with external datasets, which was done with the Netflix movie rating dataset [32].

A recently proposed way forward is given by the paradigm of federated learning [26]. Here, model fitting does not happen locally on the machines of a single party; rather, it works on distributed data without the need for central aggregation. In essence, federated learning models perform gradient descent in a distributed way,

where, instead of sharing their raw data with the server that is building the model, clients only share the gradient updates necessary to improve the loss function locally for their personal data.

While this is a promising approach, it was not originally designed with the goal of preserving privacy. Later extensions have addressed this issue, by injecting random noise into the data on the client-side before sending it to the server [3] or by using cryptography [7].

Present work: Secret vector machines (SecVM). This work proposes a novel and different approach to training a supervised machine learning classifier in a privacy-preserving manner. Crucially, in our setup, the server is assumed to be *untrusted,* i.e., potentially malicious. Our key observation is that support vector machines (SVM), a popular machine learning model, are particularly well-suited for privacy-preserving classification, due to the hinge loss function used in its objectives: when features and labels are binary (as is often the case), the SVM gradient can only take on the three discrete values -1, 0, and 1, which means particularly little information about the client is revealed.

Starting from this observation, we identify additional ingredients necessary to maintain user privacy and design a distributed protocol for training SVMs. We term our algorithm *secret vector machine (SecVM).* As in the federated learning setup [26], the SecVM model is trained on the server side in collaboration with the data-owning clients. What sets SecVM apart from prior proposals for privacy-preserving federated learning [14, 27] is that it assumes an untrusted server and works without adding random noise. Instead, it leverages the above observation regarding the SVM loss and makes the recovery of the original data from the updates shared by clients impossible by means of feature hashing, splitting updates into small chunks, and sending them to the server according to a particular communication protocol. Other popular classification algorithms such as logistic regression or neural networks do not share the property of SVM of having an integer-valued gradient, and are therefore not suited for preserving privacy in our setting.

Contributions. The main contributions of this paper are as follows.

- We propose secret vector machines (SecVM), a novel method for **training linear SVM** classifiers with integer features in a **privacy-preserving** manner (Sec. 3 and 4).
- In an **offline evaluation**, we apply SecVM to a large dataset of tweets in order to infer users' gender from the words contained in their tweets, showing that we can maintain user privacy without lowering classification performance, compared to a vanilla SVM model (Sec. 5).
- We implement SecVM's client–server setup for the Cliqz web browser and successfully deploy it in a large-scale **online evaluation** with thousands of participating clients, outperforming baselines on a gender prediction task by a large margin and thus demonstrating the feasibility of our model in production settings (Sec. 6).

By exploiting specific properties of support vector machines, our method overcomes the shortcomings of other classification models in a privacy-preserving context (cf. discussion in Sec. 7 for details). Moreover, due to their good performance on various classification tasks, SVMs are of high practical relevance for both researchers as well as industry practitioners, with recent applications in material sciences [41], geology [8], remote sensing [22] and medicine [20, 25, 39] — the latter being a particularly privacy-sensitive area. Also, although our exposition considers binary labels, it can readily be extended to the multi-class case, via schemes such as one-vs.-one or one-vs.-all [19].

2 RELATED WORK

There are two main approaches to the problem of extracting information while at the same time protecting privacy. In the first, an altered version of the data is released. In the second, all data stays on the data owners' devices, but they actively participate in the information extraction procedure.

Releasing the data. Information about individuals can be hidden by perturbing the data randomly, as applied in learning decision trees [4] and other settings [11, 13]. The notion of *k-anonymity* [35] requires each record to be indistinguishable from at least $k-1$ other records, which can be achieved by suppressing or generalizing certain attributes. Its shortcomings in terms of privacy have been addressed by *l-diversity* [24], which has further been refined to *t-closeness* [21].

Keeping the data. An area with a goal slightly different from ours is *secure multiparty computation,* where several parties each own a share of the data and want to compute a function on the full data without revealing their own data to anyone else. The problem has been solved for arbitrary functions [16], though specialized solutions for different use cases are needed for practical performance [31].

A recently proposed solution for training machine learning models on distributed data is called *federated learning* (FL) [26]: a server distributes the current version of the model to the data-owning clients, which then return only updates to this model, rather than their raw data. While FL's original focus was not on privacy, algorithms for extending it in this direction have been proposed [14, 27]. These extensions build on techniques due to Abadi et al. [3], by randomly sampling a subset of users that should participate in a training round and adding random noise to their updates. In this setting, a client's private data is not to be protected against a malicious server (the server is assumed to be trusted), but rather against other clients participating in the training. A different algorithm is based on cryptography [7], but it additionally requires a trusted public-key infrastructure via which clients can securely communicate with each other; just like the algorithm proposed in [18], which combines differential privacy with secret sharing and requires at least one of the parties among which the secret is shared to be trusted. While SecVM is similar in spirit to FL, there are distinct differences (most notably that SecVM does not assume a trusted server), which we discuss in Sec. 7.

3 PROBLEM STATEMENT

We consider the problem of training a linear SVM on distributed data, i.e., the data is spread among multiple Internet-connected client devices owned by individual parties. Each client i's dataset consists of a feature vector x_i and a label y_i. For reasons that will become clear later, we assume that features take on integer values,

which they do, e.g., if they represent counts. In addition to the clients, there is a separate party, the server owner, who would like to use the data for training a classification model. While this intention is supported by the clients, they do not want the server to learn anything about individual clients.

Why do we say "anything" above, when in fact we only want to protect *sensitive* client information? First off, the server should certainly never know the complete feature vector together with the label. However, the feature vector alone, or a part of it, could already contain sensitive information: if only a subset of the features suffices to identify a user, then the other features contain additional (potentially sensitive) information. Likewise, if a single feature is unique to a certain user, the feature–label combination gives away their label. And finally, a single feature alone can already contain sensitive information, e.g., if features are strings the user types into a text box. We therefore formalize our privacy requirement as follows:

> **Privacy requirement.** *The server must not be able to infer the label or any feature value for any individual client.*

Server capabilities. In order to infer user information, we assume a malicious server can (1) observe incoming traffic, (2) send arbitrary data to clients, and (3) control arbitrarily many clients by introducing forged clients.

4 PROPOSED SOLUTION

The loss function of a binary classification model typically takes the form

$$J(w) = \frac{1}{N}\sum_{i=1}^{N} L(w, x_i, y_i) + \lambda R(w), \tag{1}$$

where N is the number of training samples (in our case, users), $x_i \in \mathbb{R}^d$ user i's (in our case, integer) feature vector, $y_i \in \{-1, 1\}$ user i's label, $w \in \mathbb{R}^d$ the parameter vector of the model, L the loss for an individual sample, R a regularization function independent of the data, and $\lambda > 0$ the regularization parameter. When using a subgradient method to train the model, the update for dimension j becomes

$$w_j \leftarrow w_j - \eta\left(\frac{1}{N}\sum_{i=1}^{N}\frac{\partial L(w, x_i, y_i)}{\partial w_j} + \lambda\frac{\partial R(w)}{\partial w_j}\right), \tag{2}$$

where $\eta > 0$ is the learning-rate parameter. In the case of linear SVMs, we have $L(w, x_i, y_i) = \max\{0, 1 - y_i w^T x_i\}$ and

$$\frac{\partial L(w, x_i, y_i)}{\partial w_j} = \delta(1 - y_i w^T x_i)\, y_i x_{ij}, \tag{3}$$

where $\delta(x) = 1$ if $x > 0$, and $\delta(x) = 0$ otherwise.

The key observation that federated learning [26] and also our method rely on is that the update of Eq. 2 is the sum over values that each only depend on the data of a single user i. To train the model, we follow a process akin to federated learning:

(1) The server sends out the current model w to the clients.
(2) Each client i computes its local update $\nabla_w L(w, x_i, y_i)$ and sends it back to the server.
(3) The server sums up the individual updates, computes $\nabla_w R(w)$, and makes an update to the model.

To meet our privacy requirements, we adopt a slightly more nuanced protocol, described next. To begin with, we will work under the following **temporary assumptions,** which will later be dropped to accommodate real-world settings:

A1. There is a trusted third party that first collects the update data (as specified below) from the clients in every training iteration and then sends it as a single database to the server.
A2. The server honestly follows the above three-step training procedure, which essentially means that the server sends the same correctly updated weight vector to all clients.

We also assume that the client code is available for inspection by the users to make sure the clients follow the proposed protocol. The server code, however, need not be public, since SecVM protects against a malicious server.

Hiding feature values from the server: hashing. We do not want the server to learn any feature values. For this purpose, we make features less specific by grouping them via hashing. Assume we have d different features, i.e., $w \in \mathbb{R}^d$. We then choose a number $1 < k < d$ and reduce w's dimension to $\tilde{d} := \lfloor d/k \rfloor$ by applying a hash function to each feature index and taking the result modulo $\tilde{d}$ as its new index (its bin). If several features of a user hash into one bin, their values are added. In expectation, this results in at least k features, indistinguishable from each other, per bin.

Usually, hashing is used to reduce the problem dimensionality, and thereby resource consumption, and collisions are a necessary but unwanted side effect, as they may decrease model accuracy [37]. For us, on the contrary, a high number of collisions is desired, since it implies an increase in privacy: k features hashing to the same bin implies that we cannot distinguish between those k features. Lemmata 2 and 3 show that that non-colliding features, which would undermine privacy, are highly unlikely. We can, e.g., guarantee a high minimum number of collisions for each feature with overwhelming probability.

A malicious server could try to choose the hash function in such a way that certain features do not collide and can thus be distinguished. This can, however, easily be prevented by letting the hash depend on a trusted unpredictable random value, as, e.g., provided by NIST [33].

To see why we want to hide feature values from the server, consider, e.g., the case where the features are the frequencies with which strings occur in a text field into which a user types text: here, the feature itself could already reveal sensitive information, such as email addresses. Features could also help determine a user's label: if, e.g., each feature denotes whether or not a specific URL has been visited by the user, then the server could infer the label of the sole visitor i of URL j (e.g., for the URL http://github.com/User123, the sole visitor is most likely User123 themselves), by the corresponding update vector entry $\delta(1 - y_i w^T x_i)\, y_i x_{ij} = y_i$.

Hiding labels from the server: splitting update vectors. In addition to keeping the features private, we also want to prevent the server from knowing the label of a user. Recall that, during training, only the update vector $\nabla_w L(w, x_i, y_i)$ (and not the label y_i) is sent to the server. Nonetheless, in the case of a linear SVM with binary features, if one of the entries $\delta(1 - y_i w^T x_i)\, y_i x_{ij}$ of the update vector is non-zero, then that very entry equals—and

thus reveals—the label. If (and only if) the server knew to which user the update vector belongs, it would know this user's label. Since by temporary assumption A1, the server is given the update vectors only, it would have to identify the user via the update vector, which we must hence prevent. To do so, we use a property of the subgradient update (Eq. 2): not only is one user's local update independent of the other users', but also the update for one entry w_j does not rely on the update for any other entries, which means we can update each entry individually. We exploit this fact by splitting the update vector $\nabla_w L(w, x_i, y_i)$ into its individual entries and sending each entry $\delta(1 - y_i w^T x_i)\, y_i x_{ij}$ together with its index j as a separate package. In the case of binary x_{ij}, $\delta(1-y_i w^T x_i)\, y_i x_{ij}$ can only take on the values -1, 0 and 1, therefore making it impossible for the server to determine which packages stem from the same feature vector, since they cannot be discerned from other clients' packages. The case of binary features can easily be extended to integer features: instead of sending one package containing $y_i x_{ij}$, the client may send $|x_{ij}|$ packages containing $y_i \operatorname{sgn}(x_{ij})$. (Note that packages where $\delta(1 - y_i w^T x_i) = 0$ need not be sent.)

Since, after this change to the protocol, the server only receives packages containing 1 or -1 and the respective feature index, this is equivalent to knowing the number N_j^+ of positive packages and the number N_j^- of negative packages received for each feature index j. As mentioned before, $\delta(1 - y_i w^T x_i)\, y_i x_{ij} \in \{0, y_i\}$, i.e., only users with label 1 contribute to N_j^+ and only users with label -1 contribute to N_j^-. Determining the label of a user is thus equivalent to determining whether the user's update vector contributed to N_j^+ or to N_j^-. The confidence with which this can be done is vanishingly small, as we show in Lemma 4, even for the (from a privacy perspective worst) case that the server knows all of a user's features from some external source (the maximum a server that does not already know the label could possibly know).

Dropping temporary assumption A1. We now drop the assumption that the server receives the update data as a single database. In the new setting, the server additionally receives (1) the IP address from which a package was sent and (2) the time at which a package arrives. We need to make this information useless in order for the privacy guarantees from above to still hold, as follows. First, we route all messages through an **anonymity network,** such as *Tor* [38], or through a similar proxy server infrastructure, such as the one we use in our online experiment (Sec. 6), thereby removing all information that was originally carried by the IP address. Without this measure, the server could directly link all packages to the client that sent them, thus effectively undoing the above-described splitting of feature vectors. Second, to remove information that the arrival times of packages might contain, we set the length of one training iteration to n seconds and make clients send their packages not all at once, but spread them randomly over the n seconds, thus **making the packages' arrival times useless for attacks.** Without this measure, all update packages from the same user might be sent right after one another, and the server would receive groups of packages with larger breaks after each group and could thus assume that each such group contains all packages from exactly one user.

Dropping temporary assumption A2. Finally, we drop the assumption that the server honestly follows the training procedure by sending the same correctly updated weight vector to all clients in each iteration. In order to not depend on this assumption, we give clients a way to recognize when the server violates it: Instead of requesting the training data in an iteration once from the server, the clients request it multiple times. Only if they get the same response every time do they respond; otherwise they must assume an attack. Since the clients' requests are routed through an anonymization network, the server cannot identify subsequent request from the same client and cannot maliciously send the same spurious weight vector every time. To reduce communication costs, the clients don't actually request the training data multiple times, but only once in the beginning, and afterwards request hashes of it. As an even safer countermeasure, one could distribute weight vectors and all auxiliary information via a blockchain. This way, each user would be able to verify the integrity of the data they receive.

In the case that one does not prevent the server from distributing different data to different clients, the server could, e.g., distribute an all-zero weight vector $w = 0 \in \mathbb{R}^{\tilde{d}}$ to all users in a first step. All of them would then respond with a non-zero update revealing all of their non-zero features, since then $1 - y_i w^T x_i = 1$ for arbitrary y_i and x_i. In the next step, the server would send out the zero weight vector to all but one user ℓ. This user would instead receive the weight vector $e_1 = (1, 0, \ldots, 0)$. If $y_\ell x_{\ell 1} \leq -1$, then user ℓ would in this round not send back any updates. Otherwise the server would do the same with $-e_1$, then with e_2, $-e_2$, and so on, until it receives fewer updates than in the last iteration. It would then know that all the missing updates come from one user, and would thus be able to reconstruct this user's label and feature vector (however, still hashed). In a similar fashion, the server could undermine single users' privacy by manipulating the time they have for sending their updates, i.e., the length of the training iteration. All users would get a very long deadline, and one user ℓ a very short one. Then the packages arriving before the short deadline would mostly come from user ℓ.

SecVM: model and protocol. To conclude, we summarize the final SecVM model. As an initial step, all clients hash their feature vectors into a lower-dimensional space. Then the training procedure begins. The server sends out the current parameter vector w to all clients. Each client i computes its local update $g_i := \nabla_w L(w, x_i, y_i) = \delta(1 - y_i w^T x_i)\, y_i x_i$ and splits this vector into its individual entries g_{ij}. These entries, together with their indices j, are sent back to the server as individual packages at random points in time via a proxy network. The server sums up the updates g_{ij} corresponding to the same entry j of the parameter vector and updates the weight vector w accordingly. This procedure is repeated until the parameter vector has converged. This model meets the privacy requirement from Sec. 3, as it does so under temporary assumptions A1 and A2, and, as we have shown, we may drop these assumptions without violating the privacy requirement.

5 OFFLINE EVALUATION: GENDER INFERENCE FOR TWITTER USERS

We implemented and tested our approach in a real application with users connected via the Internet, as described in Sec. 6. However, to assess its feasibility and to determine suitable hyperparameters, we first worked with offline data. An and Weber [5] generously provided us with a dataset containing tweets collected from nearly 350,000 Twitter users alongside demographic data inferred from the users' profiles. Around half of the users are labeled as male, and half of them as female.

As the classification task for the SVM, we decided to predict the gender of users from the words they used in their tweets. The feature space is therefore very high-dimensional; after preprocessing, we obtained 96M distinct words. On the other hand, the feature vectors are very sparse, with only 1,826 words per user on average. The properties of the dataset and the classification task are very similar to what we would expect to encounter in our later online experiment, so we deemed this a good initial evaluation of our method.

We only compare SecVM — using different levels of hashing — with a vanilla SVM that doesn't use hashing, and not with other classification methods (logistic regression, neural networks etc.). This is due to the fact that their typically real-valued gradient entries may reveal private information, which makes those methods unsuited for fulfilling our privacy requirement (see also Sec. 7). Therefore the choice is not between training an SVM or training a different classifier, but rather between training an SVM or not being able to train a classifier at all due to the privacy constraints. Or, to put it differently: our goal is not to achieve a classification performance that can compete with non-privacy-preserving methods, but to learn a useful classifier in our restricted setting.

Methodology. For subgradient methods, one has to decide on a step size η. We chose the step size of the Pegasos algorithm [36], i.e., $\eta = 1/\lambda t$, where t is the current iteration and λ the regularization parameter; but whereas Pegasos performs stochastic subgradient training, we used regular subgradient training, where all training samples, rather than a subset, are used to compute each update.

For training and testing an unhashed SVM, we first split the dataset into two equally large sets A and B, and each of those sets into 90% training and 10% test data. Then we trained an SVM with different values of λ on A's training set, chose the one with the best accuracy on A's test set, and did another round of training on B's training set with this λ.

Results. Fig. 1a reports the performance of this second SVM on B's test set. Choosing any λ between 10^{-7} and 10^{-2} only made a marginal difference in accuracy. The subgradient method is not a descent algorithm, i.e., it does not decrease the value of the objective function in each step. The effect of this can be seen in the blue curve of Fig. 1a: test accuracy fluctuates a lot. Therefore, inspired by Corollary 1 of the Pegasos paper [36], we averaged the weight vector of subsequent iterations and investigated how the number of averaged weight vectors affects test performance. Fig. 1a shows that averaging only two weight vectors already gives not only an almost monotonic learning curve, but also a prediction accuracy strictly above that achieved when not using averaging. For the following results, we thus always averaged two subsequent weight vectors, as averaging more vectors only led to slower convergence. We thus obtained an accuracy of 75.2% on the original, unhashed data.

To evaluate the influence of hashing on accuracy, we did 141 random splits of the entire dataset into 90% training and 10% test, with fixed $\lambda = 10^{-4}$. Fig. 1b shows means and standard deviations of the fraction of correctly predicted labels for different numbers of hash bins. For instance, when reducing the dimension of the original feature and weight vectors by a factor of 1,000 (i.e., hashing the 96M original features into 96K bins), accuracy only drops from 75.2% to 74.7%, showing that even very aggressive hashing has only a negligible impact on the SVM's performance.

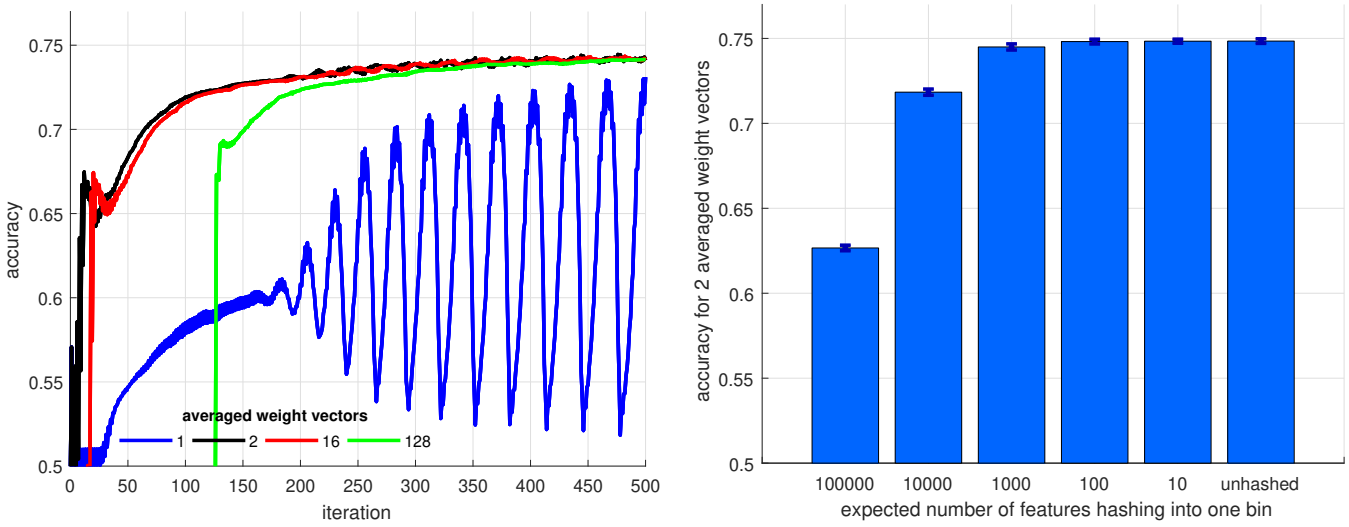
6 ONLINE EVALUATION: GENDER INFERENCE FOR WEB SURFERS

In addition to the above offline evaluation (Sec. 5), we also tested our method in its intended environment: as part of a software that runs on user-held, Internet-connected devices, in this case the Cliqz web browser [1]. Via a collaboration, we deployed SecVM to a sample of Cliqz's user base of more than 200K daily active users. For data collection, Cliqz uses the idea of client- instead of server-side data aggregation via a framework called Human Web and based on a proxy network [28, 29]. The work presented in this paper leverages some of the concepts introduced by this framework; but SecVM goes one step further and not only does data aggregation on the client side, but also computations on this data.

Task. As in the offline evaluation (Sec. 5), we decided to build an SVM that can predict a user's gender, but this time not from the words they type, but rather from the words contained in the titles of the websites they have visited. This setting is much more challenging than the one of Sec. 5: users go on- and offline at unpredictable points in time, the number of available users varies heavily between the different times of day, and together with the set of training users, the set of test users changes, too, thus giving less accurate assessments of prediction performance.

Implementation[1]**.** We implemented the client side as a module in the Cliqz browser [2]. It extracts the features (words contained in website titles) from a user's history, and the label (gender) from the HTML code of www.facebook.com once a user visits that page and logs in. The clients regularly fetch a static file from a server that describes the experiments that are currently running: the number of features to be used, the current weight vector, etc. Apart from this, it contains the percentage p of users that should participate in training, while the others are used for testing. The first time a user encounters a new experiment, they assign themselves to training with probability p, and to test with probability $1 - p$ (we chose $p = 0.7$). The file fetched from the server also informs the user how much time they have left until the end of the current training iteration to send their data back to the server. This data consists either of the update packages or a single test package, depending on whether the user is part of the training or the test set. To avoid temporal correlation attacks, each package is sent at a random point in time between its creation and the end of the training iteration.

[1]Source code available at https://github.com/cliqz-oss/browser-core/tree/6945afff7be667ed74b0b7476195678262120baf/modules/secvm/sources



(a) No feature hashing used; for various numbers of averaged weight vectors (cf. Sec. 5), each as one curve.

(b) Accuracy achieved for different numbers of hash bins (averaging two weight vectors; cf. Fig. 1a).

Figure 1: Offline evaluation on Twitter gender prediction task (Sec. 5).

IP addresses are hidden from the server by routing the packages through the Human Web proxy network [28], a system that is functionally similar to Tor and prevents identification of users at the network level.[2]

Results. We performed two runs of the experiment, one for three days during night- and daytime (Fig. 2a), and one for two days only during daytime (Fig. 2b). As parameters, we chose $\lambda = 0.1$ as the regularization parameter, 10,000 bins for feature hashing, and, as in the offline experiment on the Twitter dataset (Sec. 5), averaged two subsequent weight vectors. As can be seen in Fig. 3, there were a lot fewer females in the user base, which is why we weighed their updates by a factor of 4. We set each training iteration to last 11 minutes, in order to minimize the load on the client side.

In each iteration, we let the users from the test set evaluate the current model. We evaluate performance in terms of recall for males and females, i.e., the fraction of all males and females, respectively, that the classifier predicts correctly. Results are shown in Fig. 2a–b. The dark solid lines show recall for female and male users, respectively, the transparent lines their respective share in the test set. This share is equal to the recall that would be achieved by a baseline classifier randomly guessing each class with its marginal probability (i.e., between 10% and 20% for females, depending on the iteration number, and between 80% and 90% for males). We see that recall for males is at around the same (high) level as the baseline, while recall for females (the minority class) lies far above this baseline. However, despite the weighting, recall for females is still significantly worse than recall for males. We attribute this to the fact that the number of training samples was much smaller for this class.

[2]The technical report describing the Cliqz Human Web [28] contains a discussion on why TOR was not suitable for the Cliqz use case.

Noteworthy are the drops for females and spikes for males around iterations 150 and 300 of the continuous training (Fig. 2a). They coincide with nighttime and a drop in overall active users, as shown in Fig. 3. At night, the share of female users in the overall population drops to about half, compared to daytime (10% vs. 20%), and we explain the low female recall during those times by the fact that we see very little training signal for females then. Hence, one would preferably stop training during the night, which we did in our second run, whose results (Fig. 2b) are much more stable.

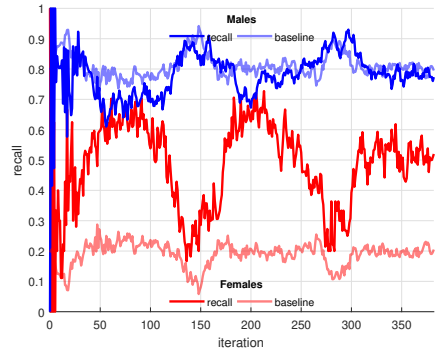
Besides recall, we also evaluated precision. It lies far above the baseline for both females and males (Fig. 2c–d).

7 DISCUSSION

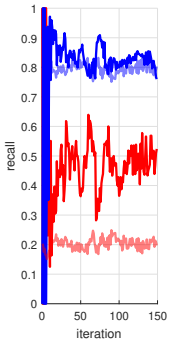
Other models. We chose the SVM as a model because of its popularity as a machine learning algorithm. This was, however, not the only reason. The SVM model also has the nice property that the partial derivatives of the user-dependent part of its loss function can only take on the values −1, 0, and 1, which we exploit to make the reconstruction of labels impossible. To illustrate this advantage, consider what would happen when using, e.g., logistic regression instead. In this case, we have $L(w, x_i, y_i) = \log(1 + \exp(-y_i w^T x_i))$ and

$$\frac{\partial L(w, x_i, y_i)}{\partial w_j} = \frac{y_i x_{ij}}{1 + \exp(y_i w^T x_i)}. \tag{4}$$

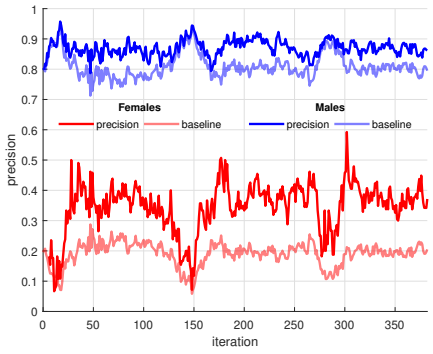
When all feature values are integers, the numerator of Eq. 4 can only take on integer values, allowing the real-valued denominator to uniquely identify user i, thus making it easy to associate updates from user i for different values of j with each other, even when splitting update vectors into their individual components, as introduced in Sec. 4 precisely as a counter-measure against such record linkage.



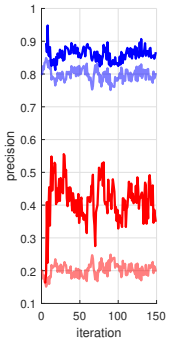
(a) ***Solid red (blue):*** **recall for females (males).** ***Transparent red (blue):*** **marginal baseline for females (males), i.e., share of females (males) in test set.**



(b) Same as Fig. 2a, but only trained during daytime.



(c) ***Solid red (blue):*** **precision for females (males).** ***Transparent red (blue):*** **marginal baseline for females (males), i.e., share of females (males) in test set.**



(d) Same as Fig. 2c, but only trained during daytime.

Figure 2: Online evaluation on task of predicting web surfers' gender (Sec. 6). We chose to perform one update every 11 minutes, so 50 updates took about 9 hours.

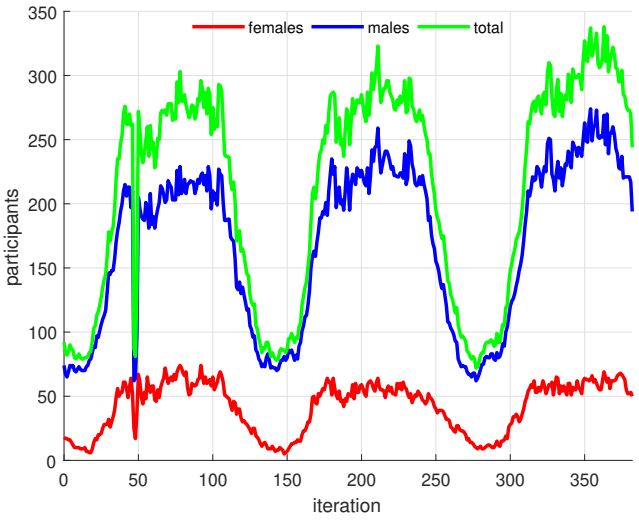


Figure 3: Online evaluation (Sec. 6): absolute number of test participants (30% of all participants) during the training shown in Fig. 2a and 2c. ***Red:*** **females.** ***Blue:*** **males.** ***Green:*** **total.**

However, we point out that rectifiers—the same function as the SVM loss—are a popular choice as an activation function for neural networks [15], and there is also research on using binary weights in neural networks [9]. Both of these observations open avenues to extend our work to neural networks, which we aim to pursue in future work.

Practical considerations. As opposed to federated learning [26], we split the updates from individual users into atomic pieces. Only this allows us to obtain such strong privacy guarantees; but, of course, it comes at the expense of efficiency: due to protocol overhead, sending many small packages instead of one big package produces more traffic both on the client and on the server side, which in turn slows down training, especially if one has no control over the client devices and has to be conservative when estimating the speed of their network connection in order not to congest it.

As opposed to the privacy-focused extensions of federated learning [14, 27], SecVM works without adding random noise. Especially when feature values are mostly small integers, such noise needs to be large in order to achieve a sufficient level of privacy. Apart from that, our goal is also a different one. The aforementioned methods [14, 27] offer differential privacy guarantees against attacks that aim at exploiting the differences between different iterations of the machine learning model during the training process that could be used by malicious clients to extract other *clients'* data, whereas the server is assumed to be trusted. SecVM, on the other hand, protects against a malicious *server*.

Often the goal of privacy research in the field of machine learning is to prevent the extraction of sensitive information from the final model. Our approach starts even earlier, by preventing the extraction of sensitive information during the training. Of course the model itself that results from such a training procedure preserves the privacy of the training data as well.

We point out that weaknesses of anonymization networks or malicious clients trying to poison the training by sending manipulated gradients are outside of the scope of this paper.

Finally, we would like to emphasize that for our privacy guarantees to hold, the client module source code must be visible to the clients, as is the case for our experiment in Sec. 6 (after review).

8 CONCLUSION

We propose SecVM, a framework for training an SVM on an *untrusted server* based on distributed data while preserving the data owners' privacy, without the need for direct access to their data. Instead, data owners only send model updates to the training server. The system relies on three main ideas to prevent the reconstruction of sensitive information from those updates: routing all messages through a proxy network, splitting the update vector into its components, and hashing the original features. We implemented SecVM

in the Cliqz browser—a real distributed system—with promising results. As a next step, we plan to extend the scheme to other machine learning models such as neural networks.

Appendices

Notation 1. *Let m denote the number of unique features and n the number of bins we hash them into. The hashing is executed by a function h drawn uniformly at random from the family of all hash functions from the set of strings to $[n]$, where by $[n]$ we denote the set $\{1, \dots, n\}$. Probabilities are taken over the random choice of h.*

We will give bounds on three probabilities that all can be used to determine the number of hash bins to use for a desired level of privacy.

Lemma 2. *(a) Let p_1 be the probability that there exists at least one feature which does not collide with any other feature. Then*

$$p_1 \leq m \left(\frac{n-1}{n}\right)^{m-1}.$$

(b) Let $K \subseteq [m]$ be a set of specific features, $k := |K|$. For the probability p_2 that at least one of the features in K does not collide with any other feature we have that

$$p_2 \leq k \left(\frac{n-1}{n}\right)^{m-1}.$$

Proof. (a) Using a union bound in the first step, we get

$$\begin{aligned} p_1 &\leq \sum_{i=1}^{n} \Pr[\text{exactly one features hashes into bin } i] \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{n} \left(\frac{n-1}{n}\right)^{m-1} \\ &= m \left(\frac{n-1}{n}\right)^{m-1}. \end{aligned}$$

In the second line we sum over over the probabilities for a specific one of the m features to end up in bin i.

(b) The statement again follows from a union bound argument:

$$\begin{aligned} p_2 &\leq \sum_{i \in K} \Pr[\text{feature } k \text{ does not collide}] \\ &= k \left(\frac{n-1}{n}\right)^{m-1} \end{aligned}$$

□

Lemma 3. *Let p_3 be the probability that each feature collides with at least $k-1$ other features. Assume that $k \leq m/n$ (otherwise $p_3 = 0$). Then*

$$p_3 \geq 1 - \binom{m}{k-1} \frac{(n-1)^{m-k+1}}{n^{m-1}} \frac{m-k+2}{m-nk+n+1}.$$

Proof. Note first that

$$\begin{aligned} p_3 &\geq \Pr[\text{at least } k \text{ features per bin}] \\ &= 1 - \Pr[\text{at least one bin with less than } k \text{ features}], \end{aligned}$$

since we exclude the possibility of having empty bins. Then by a union bound

$$\begin{aligned} &1 - \Pr[\text{at least one bin with less than } k \text{ features}] \\ &\geq 1 - \sum_{i=1}^{n} \Pr[< k \text{ features in bin } i] \\ &= 1 - \sum_{i=1}^{n} \sum_{l=0}^{k-1} \Pr[\text{exactly } l \text{ features in bin } i] \\ &= 1 - \sum_{i=1}^{n} \sum_{l=0}^{k-1} \sum_{\substack{J \subseteq [m] \\ |J|=l}} \Pr[\text{exactly the features in } J \text{ in bin } i] \\ &= 1 - \sum_{i=1}^{n} \sum_{l=0}^{k-1} \sum_{\substack{J \subseteq [m] \\ |J|=l}} \left(\frac{1}{n}\right)^{l} \left(\frac{n-1}{n}\right)^{m-l} \\ &= 1 - \sum_{i=1}^{n} \sum_{l=0}^{k-1} \binom{m}{l} \frac{(n-1)^{m-l}}{n^m} \\ &= 1 - n \sum_{l=0}^{k-1} \binom{m}{l} \frac{(n-1)^{m-l}}{n^m} \\ &= 1 - n \left(\frac{n-1}{n}\right)^{m} \sum_{l=0}^{k-1} \binom{m}{l} \left(\frac{1}{n-1}\right)^{l}. \end{aligned}$$

To bound the sum, we adapt a proof of [23]. We observe that

$$\begin{aligned} &\frac{\binom{m}{k-1}\left(\frac{1}{n-1}\right)^{k-1} + \binom{m}{k-2}\left(\frac{1}{n-1}\right)^{k-2} + \binom{m}{k-3}\left(\frac{1}{n-1}\right)^{k-3} + \dots}{\binom{m}{k-1}\left(\frac{1}{n-1}\right)^{k-1}} \\ &= 1 + (n-1)\frac{k-1}{m-k+2} \\ &+ (n-1)^2 \frac{(k-1)(k-2)}{(m-k+2)(m-k+3)} + \dots, \end{aligned}$$

which can be bounded from above by the geometric series

$$\begin{aligned} &1 + (n-1)\frac{k-1}{m-k+2} + \left((n-1)\frac{k-1}{m-k+2}\right)^2 + \dots \\ &= \sum_{l=0}^{\infty} \left((n-1)\frac{k-1}{m-k+2}\right)^{l} \\ &= \frac{m-k+2}{m-nk+n+1}. \end{aligned}$$

The series converges because $k \leq m/n$. This calculation yields

$$\begin{aligned} &1 - n \left(\frac{n-1}{n}\right)^{m} \sum_{l=0}^{k-1} \binom{m}{l} \left(\frac{1}{n-1}\right)^{l} \\ &\geq 1 - n \left(\frac{n-1}{n}\right)^{m} \binom{m}{k-1} \left(\frac{1}{n-1}\right)^{k-1} \frac{m-k+2}{m-nk+n+1} \\ &= 1 - \binom{m}{k-1} \frac{(n-1)^{m-k+1}}{n^{m-1}} \frac{m-k+2}{m-nk+n+1}. \end{aligned}$$

□

To get some intuition for these quantities, we give an example. In the Twitter experiment in Sec. 5 with $m = 95,880,008$ we saw no significant decrease in prediction accuracy for $n = 95,880$. For these values, we get $p_1 < 5 \times 10^{-427}$ and $p_2 < 6k \times 10^{-435}$. The k in Lemma 3 can be chosen between 1 and 1,000. For $k = 700$, we have $1 - p_3 < 5 \times 10^{-19}$.

In [37] it was proved that hashing each feature into multiple bins can increase the probability that for each feature there exists at least one bin where it does not collide with any other feature. Of course this only holds if the number of bins is higher than the number of features; also we don't want features that have no collisions. Nevertheless, this is an interesting option even in our case. Assume that the number of features is higher than the number of bins, but that many of them are not very indicative of the label we are trying to predict. Thus we are only interested in preventing collisions between indicative features. If we set n in Theorem 1 of [37] to be the number of indicative features, we obtain a bound for their non-collision probability if we hash all features into multiple bins. In our experiments in Sec. 5 and Sec. 6 we are in a situation where most features are not very indicative — however, increasing the number of bins each feature is hashed into slightly decreased the accuracy instead of increasing it.

As shown in Sec. 4, determining whether a user's label is 1 (-1) is equivalent to determining whether they have contributed to the sum of positive (negative) update vectors. For this task, we allow the server the maximal knowledge, i.e., the knowledge of the entire feature vector, which equals the update vector up to multiplication with -1, 0 or 1.

We formalize the task of the server as the discrimination between two worlds. In world 1, the vectors of all users are random. In world 2, the vectors of all users except from the one to be attacked are random; the vector of the latter one is known to the server.

Lemma 4. *Let u_j denote the j-th entry of a vector $u \in \mathbb{N}^d$. Let $M > 1$ be the number of users participating in the training, $d > 0$ the dimension of the update vectors and $F > 0$ the (fixed) ℓ_1-norm of the update vectors. The model is trained for $K > 0$ iterations.*
Let $X^{mkf} \sim U(u \in \mathbb{N}^d_{\geq 0} :$ There exists exactly one $j^* \in \{1,\dots,d\}$ s.t. $u_{j^*} = 1$ and $u_j = 0$ for all $j \neq j'$) be i.i.d. random vectors following the uniform distribution over all one-hot vectors, where $m \in \{1,\dots,M\}$, $k \in \{1,\dots,K\}$, $f \in \{1,\dots,F\}$. The update vector of the m-th user in iteration k is then given by $X^{mk} := \sum_{f=1}^{F} X^{mkf}$. (Here we assume positive updates; the case of negative updates is analogous.) Let $v^k \geq 0$, $\|v\|_1 = F$, be the (possible) update vector in the k-th training iteration of the user to be attacked. Further let $s^k \in \mathbb{N}^d_{\geq 0}$, $\|s^k\|_1 = MF$, be the sum that the server receives in the k-th iteration. Then

$$
\begin{aligned}
&|\Pr[\sum_{m=1}^{M} X^{mk} = s^k \quad \textit{for all } k = 1 \dots K] \\
&\quad - \Pr[\sum_{m=1}^{M-1} X^{mk} + v^k = s^k \quad \textit{for all } k = 1 \dots K]| \\
&\leq \max\{p(M,F,d),\ p(M-1,F,d)\}^K,
\end{aligned}
$$

where

$$
p(m,f,d) = \frac{(mf)!}{\left(\lfloor \frac{mf}{d} \rfloor !\right)^d} \frac{1}{d^{mf}}.
$$

Proof. We show that both probabilities are bounded by the r.h.s. and then use that $|a - b| \leq \max\{a,\ b\}$ for non-negative a and b.

$$
\begin{aligned}
&\Pr[\sum_{m=1}^{M} X^{mk} = s^k \quad \text{for all } k = 1 \dots K] \\
&= \Pr[\sum_{m=1}^{M} X^{m1} = s^1]^K \\
&= \Pr[\sum_{m=1,f=1}^{M,F} X^{m1f} = s^1]^K \\
&= \Pr[\sum_{m=1,f=1}^{M,F} X_i^{m1f} = s_i^1 \quad \text{for all } i = 1 \dots d]^K \\
&= \left(\binom{MF}{s_1^1, \dots, s_d^1} \frac{1}{d^{MF}} \right)^K \\
&\leq \left(\frac{(MF)!}{\left(\lfloor \frac{MF}{d} \rfloor !\right)^d} \frac{1}{d^{MF}} \right)^k \\
&= p(M,F,d)^K
\end{aligned}
$$

For the second probability we similarly obtain

$$
\begin{aligned}
&\Pr[\sum_{m=1}^{M-1} X^{mk} + v^k = s^k \quad \text{for all } k = 1 \dots K] \\
&= \left(\binom{(M-1)F}{s_1^1 - v_1^1, \dots, s_d^1 - v_d^1} \frac{1}{d^{(M-1)F}} \right)^K \\
&\leq p(M-1,F,d)^K,
\end{aligned}
$$

where we tacitly assumed that $s_i^k - v_i^k \geq 0$ for all i and k; otherwise

$$
\Pr[\sum_{m=1}^{M-1} X^{mk} + v^k = s^k \quad \text{for all } k = 1 \dots K] = 0.
$$

□

In the lemma we assume that all user have the same amount of features (as is the case in our online experiment). However, we only use this for simplifying the notation; one can also do without this assumption and get a corresponding bound. One simply has to replace MF by the total number of features and $(M-1)F$ by this number minus $\|v\|_1$ in the probability bounds.

Taking $M = 34,615$, $F = 1826$, $d = 95,880$ — the numbers from the Twitter experiment if we pretend to have the same number of features for each user and assume that 10% of the users send a positive update —, the lemma gives a probability bound of $< 5 \times 10^{-173411}$.

REFERENCES

[1] 2018. Cliqz. https://cliqz.com/.

[2] 2018. SecVM Client. https://github.com/cliqz-oss/browser-core/tree/b5873bfaccbe67a3ebf76dbc9baf24900056cb86/modules/secvm/sources.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[4] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-preserving data mining. In *ACM Sigmod Record*.

[5] Jisun An and Ingmar Weber. 2016. #greysanatomy vs. #yankees: Demographics and Hashtag Use on Twitter. In *Proceedings of the Tenth International Conference on Web and Social Media, Cologne, Germany, May 17-20, 2016*. 523–526. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM16/paper/view/13021

[6] Michael Barbaro and Tom Zeller Jr. 2006. A Face Is Exposed for AOL Searcher No. 4417749. http://www.nytimes.com/2006/08/09/technology/09aol.html.

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1175–1191.

[8] Wei Chen, Hamid Reza Pourghasemi, Aiding Kornejady, and Ning Zhang. 2017. Landslide spatial modeling: Introducing new ensembles of ANN, MaxEnt, and SVM machine learning techniques. *Geoderma* 305 (2017), 314 – 327. https://doi.org/10.1016/j.geoderma.2017.06.020

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.

[10] Department of Homeland Security. 2014. Increase in Insider Threat Cases Highlight Significant Risks to Business Networks and Proprietary Information. http://www.ic3.gov/media/2014/140923.aspx.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*. Springer, 265–284.

[12] European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union* L119 (4 May 2016), 1–88. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC

[13] Alexandre Evfimievski. 2002. Randomization in Privacy Preserving Data Mining. *SIGKDD Explor. Newsl.* 4, 2 (Dec. 2002), 43–48. https://doi.org/10.1145/772862.772869

[14] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially Private Federated Learning: A Client Level Perspective. *arXiv preprint arXiv:1712.07557* (2017).

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 315–323.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 218–229.

[17] Glenn Greenwald and Ewen MacAskill. 2013. NSA Prism program taps in to user data of Apple, Google and others. http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[18] Mikko Heikkilä, Eemil Lagerspetz, Samuel Kaski, Kana Shimizu, Sasu Tarkoma, and Antti Honkela. 2017. Differentially private Bayesian learning on distributed data. In *Advances in Neural Information Processing Systems*. 3226–3235.

[19] Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks* 13, 2 (2002), 415–425.

[20] Min-Wei Huang, Chih-Wen Chen, Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. 2017. SVM and SVM Ensembles in Breast Cancer Prediction. *PLOS ONE* 12, 1 (01 2017), 1–14. https://doi.org/10.1371/journal.pone.0161501

[21] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 106–115.

[22] Peng Liu, Kim-Kwang Raymond Choo, Lizhe Wang, and Fang Huang. 2017. SVM or deep learning? A comparative study on remote sensing image classification. *Soft Computing* 21, 23 (01 Dec 2017), 7053–7065. https://doi.org/10.1007/s00500-016-2247-2

[23] Michael Lugo. 2017. Sum of 'the first k' binomial coefficients for fixed n. MathOverflow. https://mathoverflow.net/q/17236 (version: 2017-10-01).

[24] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. 2006. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 24–24.

[25] Sahila Mohammed Marunnan, Babitha Pallikkara Pulikkal, Anitha Jabamalairaj, Srinivas Bandaru, Mukesh Yadav, Anuraj Nayarisseri, and Victor Arokia Doss. 2017. Development of MLR and SVM Aided QSAR Models to Identify Common SAR of GABA Uptake Herbal Inhibitors used in the Treatment of Schizophrenia. *Current neuropharmacology* 15, 8 (November 2017), 1085—1092. https://doi.org/10.2174/1567201814666161205131745

[26] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*. http://proceedings.mlr.press/v54/mcmahan17a.html

[27] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private language models without losing accuracy. *arXiv preprint arXiv:1710.06963* (2017).

[28] Konark Modi, Alex Caterineu, Philipe Classen, and Josep M. Pujol. 2017. *Human Web Overview*. Technical Report. Cliqz.

[29] Konark Modi and Josep M. Pujol. 2015. Collecting User's Data in a Socially-Responsible Manner. In *European Big Data Conference*. Linux Foundation.

[30] Elen Nakashima. 2007. Verizon Says It Turned Over Data Without Court Orders. http://www.washingtonpost.com/wp-dyn/content/article/2007/10/15/AR2007101501857_pf.html.

[31] Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 448–457.

[32] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 111–125.

[33] NIST. 2017. NIST Randomness Beacon. https://www.nist.gov/programs-projects/nist-randomness-beacon.

[34] Andrea Peterson. 2015. Bankrupt RadioShack wants to sell off user data. But the bigger risk is if a Facebook or Google goes bust. Washington Post, March 26th 2015.

[35] Pierangela Samarati and Latanya Sweeney. 1998. *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*. Technical Report. Technical report, SRI International.

[36] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming* 127, 1 (01 Mar 2011), 3–30. https://doi.org/10.1007/s10107-010-0420-4

[37] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, Alex Strehl, and Vishy Vishwanathan. 2009. Hash kernels. In *Artificial Intelligence and Statistics*.

[38] Paul Syverson, R Dingledine, and N Mathewson. 2004. Tor: The secondgeneration onion router. In *Usenix Security*.

[39] L. Wang, P. C. Pedersen, E. Agu, D. M. Strong, and B. Tulu. 2017. Area Determination of Diabetic Foot Ulcer Images Using a Cascaded Two-Stage SVM-Based Classification. *IEEE Transactions on Biomedical Engineering* 64, 9 (Sept 2017), 2098–2109. https://doi.org/10.1109/TBME.2016.2632522

[40] Wikipedia. 2018. List of data breaches. https://en.wikipedia.org/w/index.php?title=List_of_data_breaches&oldid=825018241.

[41] D S Ye, Y H J Fuh, Y J Zhang, G S Hong, and K P Zhu. 2018. Defects Recognition in Selective Laser Melting with Acoustic Signals by SVM Based on Feature Reduction. *IOP Conference Series: Materials Science and Engineering* 436, 1 (2018), 012020. http://stacks.iop.org/1757-899X/436/i=1/a=012020