# Succinct ZK proof for Machine learning classifiers

Sanket Kanjalkar

smk7@illinois.edu

Yunqi Li

yunqil3@illinois.edu

## 1 Motivation

In most of the cryptocurrencies like Bitcoin, the details of a transaction, sender, receiver and amount transferred, are public. Privacy such as anonymity and confidentiality need to be improved for more extensive use of blockchain. Even though Bitcoin provides some weak anonymity though the unlinkability of Bitcoin addresses and real world identities, it still lacks of confidentiality. In order to provide the confidentiality of transaction amounts, confidential transactions(CT) are introduced. A Zero-Knowledge proof needs to be included to prove the validity of each CT. Current proposals for CT Zero-Knowledge proofs have either been prohibitively large(STARK) or required a trusted setup(SNARK). In this way, Bulletproofs is proposed to provide short non-interactive Zero-Knowledge proofs without a trusted setup.

Recently, Bulletproofs have some hype in social media. Monero, a privacy based cyrptocurrency has implemented bullet proofs which they claim has reduced the size of transactions by about 85 percent. Because of trustless setup of bulletproofs, we are excited about it's impact on blockchain space and thus we would like to explore the following as a part of this course project.

## 2 Background

### 2.1 Linear Classifier

Given a set of points in $d$-dimensional hyperspace, each marked as belonging to one or the other of two categories, an SVM (Support Vector Machine) training algorithm builds a model that assigns new examples to one category or the other. This algorithm creates a straight line which separates data points into 2 sections. Figure 1 shows three possible candidates for classifiers $H1, H2, H3$. $H3$ is a optimal classifier whereas $H2$ and $H1$ have some inaccuracies.

### 2.2 SVM Algorithm

Examples in dataset are in the form of $(\vec{x_i}, y_i)$, where $\vec{x_i}$ is $d$-dimensional vector and $y_i$ is ether $-1$ or 1 denoting which category $\vec{x_i}$ belongs to.

In the case of SVM, we want to find a linear classifier, which is actually a $(d-1)$-dimensional hyperplane. Any hyperplane can be written as

$$\vec{w} \cdot \vec{x} - b = 0$$

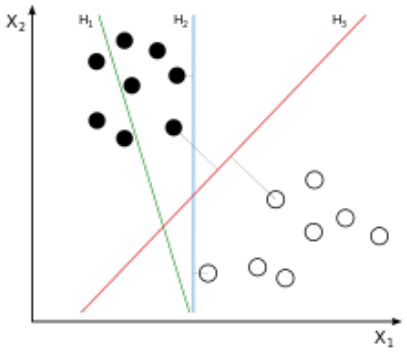



Figure 1: Linear classifiers.

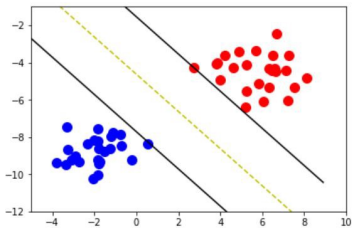


Figure 2: SVM algorithm

In this scenario, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin" (Ref Figure 2), and the maximum-margin hyperplane is the hyperplane that lies halfway between them. The distance between these two hyperplanes is $\frac{2}{\|\vec{w}\|}$. In order to maximize the distance, we need to minimize $\|\vec{w}\|$. We also have to prevent data points from falling into the margin, which means the following constraints needs to be satisfied(Ref Figure 2):

$$\vec{w} \cdot \vec{x_i} - b \leq 1, \ \texttt{if} \ y_i = 1$$

or

$$\vec{w} \cdot \vec{x_i} - b \geq -1, \ \texttt{if} \ y_i = -1$$

for all $i \in \{1, 2, \ldots, n\}$. This could be written as

$$y_i \cdot (\vec{w} \cdot \vec{x_i} - b) \geq 1$$

For prediction, we classify a **new** point $\vec{x}$ via

$$sgn(\vec{w} \cdot \vec{x} - b)$$

### 2.3 Note: From SVM to deep learning

Although there are optimal deterministic solutions to completely mathematically solve the linear classifier for a linearly separable dataset, there are no known optimal solutions for a more complicated dataset. The field of Deep learning uses a combination of these linear classifiers(called perceptron) to classify in a more complex fashion.

## 3 Problem Description

### 3.1 Notations, formulations and assumptions

#### 3.1.1 Notations

$\vec{x_i}, y_i$ denotes the training data points which are assumed to distributed already to all participants(sellers and 1 customer). $\vec{x_i}$ denotes the attributes, whereas $y_i$ denotes the label
$\vec{w}$ denotes the secret value or witness which is the classifier in this case
$A, B$ capital values denote points on the Eliptic curve(assumed to be secp256k1)
$r, v$ denote scalar values inside the field of secp256k1

### 3.1.2 Assumptions

We assume that all models trained are over integer fields to avoid representing floating point numbers.
We represent negative numbers by implicitly assuming numbers above certain range as negative in the field.

## 3.2 Description

### 3.2.1 Setup phase

Suppose a customer has a large dataset denoted by $\vec{x_i}$ where $1 \leq i \leq n$ on which they want to train a classifier. The customer would like to buy a classifier from multiple possible sellers. In a Byzantine scenario, the sellers can cheat by claiming that their model has higher accuracy than it actually does. On the other hand, the sellers also don't want to reveal any information about the classifier unless the customer buys it. This scenario provides a good candidate where Zero knowledge might help convince the customer that the customer indeed has the solution without revealing any information about the solution.

### 3.2.2 Deal Phase

In this phase, all sellers give a ZK proof of their classifier to the customer, and customer calculates the accuracy of classifiers. Based on accuracies, customer can make a decision which seller should he buy the classifier from.

Aside: In a blockchain setting, the buying and transfer of secret can also be be made atomic using Adaptor signatures, but that is out of the scope of this project.

## 3.3 Simple ZK Proof Scheme

All of our calculations are done in $\mathbb{Z}_p$. If $\vec{w} \cdot \vec{x} - b < 0$, then this value will overflow. Since the order of the group in Elliptic Curves is a 256-bit number and extremely close to $2^{256}$, suppose for any positive number $v \in [0, 2^{255})$ and all negative values fall into the interval $[2^{255}, p)$. Without loss of generality, we only provide Zero-Knowledge proof for $v \in [0, 2^{255})$, as the other case is in a similar way.

The customer will evaluate each trained model according to its accuracy, i.e. the number of examples predicted correctly. For each example $(\vec{x_i}, y_i)$, assume $y_i' = \vec{w} \cdot \vec{x_i} - b$. The seller will provide a Zero-Knowledge proof for $y_i' > 0$ as following:

$$\{(\vec{w} \in \mathbb{Z}_p^d, b \in \mathbb{Z}_p) : g^{\vec{w} \cdot \vec{x_i}} = A,\ g^b = B,\ g^v = V,\ v = \vec{w} \cdot \vec{x} - b,\ 0 \leq v < 2^{128}\}$$

Next, we use Pedersen commitment to provide the perfectly hiding property. The Zero-Knowledge proof with commitment scheme is as following:

$$\{(\vec{w} \in \mathbb{Z}_p^d, b, r_w, r_b, r_v \in \mathbb{Z}_p) : g^{\vec{w} \cdot \vec{x_i}} h^{r_w} = A,\ g^b h^{r_b} = B,\ g^v h^{r_v} = V,\ v = \vec{w} \cdot \vec{x} - b,\ 0 \leq v < 2^{128}\}$$

## 3.4 Optimization by Bulletproofs

The critical contribution of Bulletproofs is introducing an inner-product argument with communication complexity $2\log_2(n)$. And it also improves on the linear sized range proofs greatly. With Bulletproofs, we can improve our Zero-Knowledge proof for SVM algorithm.

Suppose $\vec{w} = (w_1, \ldots, w_d)$ and $\vec{x_i} = (x_1, \ldots, x_d)$. Let $\vec{w'} = (w_1, \ldots, w_d, b)$ and $\vec{x_i'} = (x_1, \ldots, x_d, -1)$. In this way,

$$v = \vec{w} \cdot \vec{x_i} - b = \vec{w'} \cdot \vec{x_i'}$$

The version improved by Bulletproofs is as following:

$$\{(\vec{w'} \in \mathbb{Z}_p^d, r_{w'}, r_v \in \mathbb{Z}_p) : g^{\vec{w'} \cdot \vec{x_i'}} h^{r_{w'}} = W,\ g^v h^{r_v} = V,\ v = \vec{w'} \cdot \vec{x_i'},\ 0 \leq v < 2^{128}\}$$

Moreover, with Bulletproofs, one can prove that $m$ commitments lie in a given range by providing only an additive $O(\log(m))$ group elements over the length of a single proof. We can construct a aggregated proof for all $x_i$ that further optimizing our Zero-Knowledge proof.

# 4 Implementation

## 4.1 Dataset

The dataset we used is a subset of MNIST. Each example is a picture of number from 0 to 9 as shown in the Figure 3 below. To fit it in our binary classification task, we separate them into odd and even numbers. The full image data is stored in a $784 \times 70000$ matrix and the corresponding labels are stored in a 70000 element array.

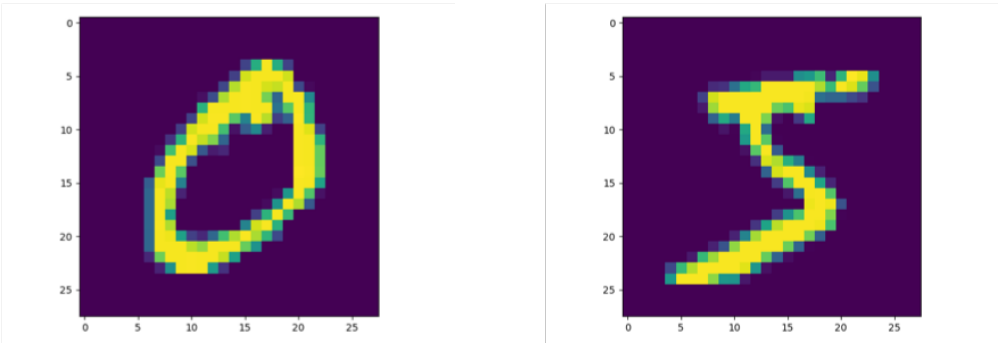


Figure 3: dataset sample

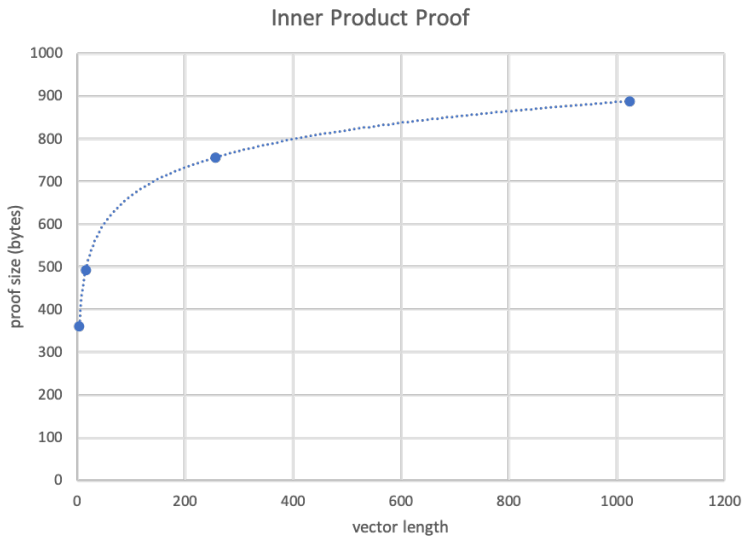
## 4.2 Transformation Accuracy Loss

| scaling parameter | 1 | 10 | $10^2$ | $10^3$ - $10^{16}$ | before conversion |
|---|---|---|---|---|---|
| Score | 0.5225 | 0.893 | 0.899 | 0.8975 | 0.8975 |

Table 1: Loss of Accuracy

Usually, all data used in svm function are floating-point numbers. We need to convert those float number to integer field elements to implement zero-knowledge proof. During the transformation, we convert the corresponding decimal to fixed point decimal and truncate to the nearest integer. For example, we will convert 0.124872 to 1248 with scaling parameter $10^4$.

Table 1 shows the score with different scaling parameters. Once the scaling parameter is larger than $10^2$, the truncation won't interfere with the accuracy, which means this transformation is practical.




### 4.3 Experiment Results

The following figures and tables show the size of bulletproof as the increasing of witness size. We can see that the growth is logarithmic. Whenever the length of input vector double, the proof size will increase exactly 66 Bytes(2 points on ecliptic curve 2*33). As we have 2 bulletproofs, our proof size increases by $33 * 4 = 112$ bytes for doubling vectors.
One bullet proof for proving that a $y_i$ is a certain range. This serves as a check for number of being positive or negative. The threshold for this is denoted by $RangeBits$ in the table.
And the second bullet proof for checking whether the inner product $\vec{w'} \cdot \vec{x'_i}$ is computed correctly. This is standard inner product argument as presented in the paper.

| vector length | 4 | 16 | 256 | 1024 |
|---|---|---|---|---|
| size(bytes) | 361 | 493 | 757 | 889 |

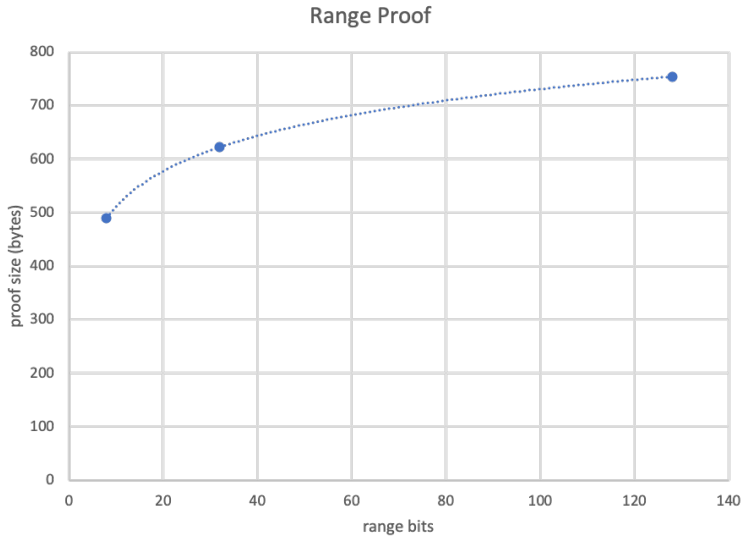
Table 2: Inner Product Proof Size

| Range Bits | 8 | 32 | 128 |
|---|---|---|---|
| size(bytes) | 490 | 622 | 754 |

Table 3: Range Proof Size

Table 4 shows the sum of generation and verification time with different input size. And Table 5 shows the comparison of naive Zero-Knowledge proof, pure BulletProofs, BulletProofs with optimizations (aggregation and batch verification).

Our code and running instruction is public on GitHub.




| RangeBit<br>VectorLen | 8 | 32 | 128 |
|---|---|---|---|
| 4 | 1.91456890106 | 7.1203892231 | 23.047246933 |
| 16 | 3.15749692917 | 8.08064985275 | 24.6021609306 |
| 256 | 28.2367072105 | 31.8720450401 | 49.6011371613 |
| 1024 | 108.965355873 | 109.09849 | 122.272001982 |

Table 4: Time of Generation and Verification

| Implementation | Size in bytes | Time of execution | Formula<br>(n elements; m bits each) |
|---|---|---|---|
| Naïve OR proofs | 380Kb | $\sim$ 20.1 mins | size=O($n \times m$)<br>time = O($n \times m$) |
| Bullet proofs | $1643 \times 10 = 15$Kb | = 20.1 mins | size=O($n\times$ log(m))<br>time = O(n$\times$m) |
| Bullet proofs:<br>Aggregation | $1643 + 33 \times 20 = 2.3$Kb | $\sim$ 20.1 mins | size=O(log(nm))<br>time = O(n$\times$m) |
| Bullet proofs:<br>Aggregation +<br>Batch verification | $1643 + 33 \times 20 = 2.3$Kb | – | size=O(log(nm))<br>amortized-time = O(m) |

Table 5: Comparison of Different Proofs