# ZK-friendly ML model explorations - Milestone 4

Saeyoon Oh

March 2024

This article outlines the progress of milestone 4 for acceleration program by Ethereum PSE team. Article first restates the goals made for the report, concluding with the progress made.

## 1 Goals

The goal of milestone 4 is to provide source code and documentations of how one can make classification using k-means clustering given heart failure dataset. We provide source code to prove the result of k-means clustering based classification using EZKL and Circom.

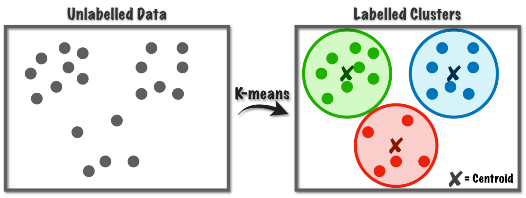
- Functionality: Provide source code to train and prove k-means clustering based classification using EZKL.
- Functionality: Provide approach to prove k-means clustering based classification using Circom.
- Analysis: Analyze the number of constraints required for proving k-means clustering based classification, compare them with neural network and decision tree.

## 2 k-means clustering

Unlike previous machine learning methods (neural network (NN) and decision tree (DT)), k-means clustering is an unsupervised machine learning algorithm. Specifically, it aims to cluster the data such that each cluster represents a homogeneous class.

Since k-means clustering is an unsupervised approach, we do not explicitly train the model parameters. k-means clustering initially set points (reference points) to represent each class. Data points are iterated and are classified to the closest reference points. After all data have been classified, new reference points are sampled by calculating the center of each class.

The cycle of choosing new reference points is called an iteration. Aforementioned iteration repeats unless predefined condition is met or the iteration number exceeds the preset value.



For heart failure classification, we make two clusters using the given data points. After clustering algorithm has ended, we retrieve the two reference points. When a new data comes in, the classification is made by calculating the L2 distance between the new data and the two reference points. If given data is closer to reference point of heart failure cluster, the data is classified to have the heart failure, and vice versa.

## 3 Progress

The progress made are as follows.

### 3.1 Proving k-means clustering Classification: EZKL

We first provide source code for k-means clustering the heart-failure dataset. Users can run k-means clustering by running the following command :

```
$ python train_kmeans.py
```

If the data instance lies within the first cluster, the data turns out to be non heart-failing and vice versa. After clustering, the accuracy is reported by running the test dataset. The clustering result is shown in Figure 3.1 where the red dots are heart-failure data instances. Since the instances have feature dimension 18, we plot the points after performing PCA to reduce the dimension to 2.

After that, you can prove the classification result by running:

```
$ python prove_kmeans_ezkl.py
```

The source code runs the inference of test dataset using the saved k-means clustering model and proves the inference was actually made by the model. We note that specific *max_logrows* and *scales* parameters should be given in order to generate the proof. We use value of 20 and 1 for each parameters. If the two parameters are not set properly, low *max_logrows* for instance, EZKL fails the *calibration* step and the proof generation fails. Yet, we were not able to fully investigate this failure.

The time analyzed for proof generation and analysis are as follows:

- setting_time: 0.02
- calibration_time: 0.38

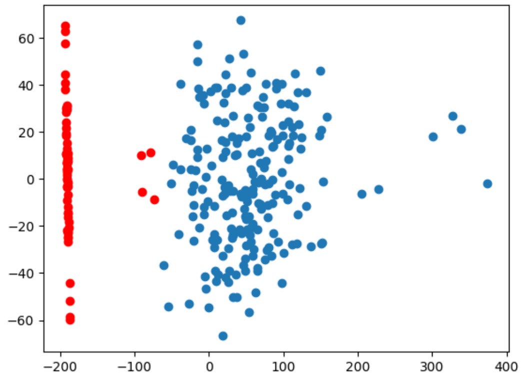


Figure 1: Cluster Visualization

- compile_time: 0.0018
- get_srs_time: 0.32
- witness_generation_time: 0.21
- setup_time: 95.72
- proof_generation_time: 244.73
- verification_time: 0.23
- Accuracy: 59.42
- ZK Accuracy: 59.42

Although the inference using k-means clustering takes the simplest form compared to neural networks and decision tree, the proof generation with EZKL takes the most amount of time. It is also noteworthy to mention that the setup time is much larger than decision tree of depth 3 (4.975 seconds) and MLP with 1 hidden layer and with hidden dimension of 8 (28.38 seconds). We note here that the number is different from milestone report 1. This is while previous proving program was using batch size of 16, requiring multiple iterations, we use batch size of 276 (whole test dataset) to match settings with k-means clustering and decision tree.

While the protocol for k-means clustering is much simpler than that of decision tree or neural network, the proof generation time tends to be larger. Having experienced a requirement to fine-tune parameters in the setup phase, we conjecture that this is a EZKL-specific behavior, in which the project is optimized for more genrally-used neural networks.

Another important thing to note for k-means clustering is that it does not require sensitive scaling. Since the model does not require any parameters usually in floating point format, (neural network, for instance) the algorithm just involves calculating reference points which relies in the quantitative level as features. Therefore scaling is not necessary for datasets that have high (larger than 1) values. Also even if the dataset features are in range of [0, 1], the dataset itself can be scaled to remove scaling dependency when proving. This results in no performance degradation when converted to ZK circuit.

## 3.2 Proving k-means clustering Classification: Circom

We also provide pipeline for proving k-means clustering using Circom. As mentioned before, proving classification using k-means clustering is consisted of two steps. First one needs to calculate the L2 distance between each reference points and the data point that one wants to inference about. Next, one needs to compare the L2 distances and make prediction out of it.

We build Circom circuit kmeans.Circom that proves these two operations. We allow batched inference (Inferencing multiple instances.) in order to compare the time results with other methods. Users need to provide the pre-calculated reference points along with the data instances that users want to inference. The circuit then proves by following the two aforementioned processes.

You can again get pre-calculated reference points by running:

```
$ python train_kmeans.py
```

which saves the reference points, and prove classification by running:

```
$ cd Circom_data
$ bash key-gen.sh
$ cd ..
$ python prove_kmeans_ezkl.py
```

Users may want to change the filename inside *key-gen.sh* since the file is also used for Circom circuit commitment for neural networks.

The time analyzed for proof generation and analysis are as follows:

- witness_generation_time: 1.01
- proof_generation_time: 5.10
- verification_time: 3.77
- Accuracy: 59.42
- ZK Accuracy: 59.42

The proof generation time is smaller that of MLP (1 hidden layer, hidden dimension of 8) which is 138.87. Here, unlike result in EZKL we use the previous setting where proof generation is done in batched iteration manner with batch size of 16. Such decision was made since increasing the batch size to 276 caused the circom circuit compilation and key generation setup to take too much time. While it is slower than that of decision tree of depth 3, (1.142) we again emphasize that the decision tree algorithm used for Circom proof generation is limited (details in milestone report 3).

Also the Circom exhibits the same characteristic that it is not dependant on scaling factors due to aforementioned data-specific protocol.

## 3.3 Conclusion

In this paper, we report the time required for proving k-means clustering based classification and compare them with neural networks and decision tree. For EZKL, the proof generation tends to be longer than the two other methods, which we believe is EZKL-specific matter (less optimized). For Circom, this results in faster proof generation compared to neural network, due to its small computational usage and no use of activation function (ReLU).

Throughout the reports, we compared EZKL and Circom on three different kinds of model types; neural networks, decision trees, and k-means clustering. While the average proof generation time tends to be shorter for Circom, this may result from the use of look ups (nonlinear table) on certain parts of the model. We were able to observe the use of nonlinear columns for cases that do not involve the use of nonlinear functions, in which we conjecture the EZKL protocol is internally using for specific low-level operations.

Yet, EZKL provides much greater flexibility and usability. They provide automatic calibrations and great interface that allows user to efficiently change scaling factors (which must be done by hand for Circom). Also Circom proof generation does not involve lookups, which prohibits the use of nonlinear activation functions that are widly used in a lot of deep learning models. Also, Circom requires manually crafting each circuits while such job is not required for EZKL.

Our future works include implementing full decision tree proof generation using Circom, and investigating the proof generation size in *matched FLOPS setting*. We believe that although matching the FLOPS would require fine grained construction of each model, it would provide more capability to analayze how efficiently each model can be proven upon the exact same machine-level computational complexity.